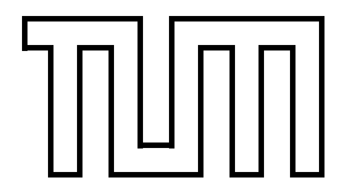


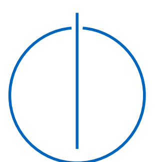
DEPARTMENT OF INFORMATICS

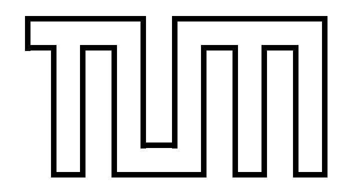
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Design and Evaluation of a Collaborative Approach for API Lifecycle Management

Duc Huy Bui





DEPARTMENT OF INFORMATICS

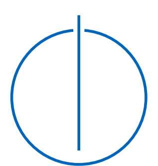
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Design and Evaluation of a Collaborative Approach for API Lifecycle Management

Design und Evaluation eines kollaborativen Ansatzes für API Lifecycle Management

Author:	Duc Huy Bui
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Gloria Bondel
Submission Date:	October 15, 2018



I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, October 15, 2018 Duc Huy Bui

Acknowledgments

I would first like to thank my thesis advisor M.Sc. Gloria Bondel from the Department of Informatics at the Technical University of Munich, Germany. In addition, I also want to thank my second advisor M.Sc. Dennis Seidel from the industry partner. Both were always open whenever I ran into a trouble spot or had a question about my research or writing. I want to thank both for their helpful constructive feedback and steering me in the right direction.

I also want to thank my supervisor Prof. Dr. Florian Matthes for providing me this thesis' topic and the opportunity to write this thesis in cooperation with an industry partner. Furthermore, I want to thank him for his valuable, constructive feedback and helpful discussions.

I would also like to acknowledge M.Sc. Ömer Uludag who helped me to establish the connection to the industry partner and without him, I would not have been able to conduct this research in the current setup.

My further thanks goes to my colleague Dr. Matheus Hauder for giving me further useful tips for my research. I am gratefully indebted to his very valuable comments on this thesis.

Further gratitude is given to all industry experts and researchers who were involved in the evaluation interview for this research project. Without their passionate participation and input, the evaluation could not have been successfully conducted.

Finally, I must express my very profound gratitude to my family, my girlfriend and my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

In recent years, Application Programming Interfaces (API) gained significantly high attention from many companies. Companies discovered the emerging trend of the API economy where they can monetize their APIs by giving users and partners access to their back-end functionalities and data properties via APIs. By making APIs accessible to external or partner consumers, companies are able to reach new markets, enable their business strategy and drive the creation of new innovative solutions.

However, APIs are often not well designed or do not meet sufficient quality standards which lead to missing business opportunities. Moreover, the API provider needs to include external users more into the API development process, since collaboration is inherently needed to scale the API program. For securing the competitive advantage, business growth as well as the innovation potential of a company, API Management will play a crucial role, especially the Full Lifecycle API Management.

The goal of this research is to find a way to assist the API Lifecycle process that is driven by collaboration. Therefore this thesis proposes an approach to design and implement a Collaborative API Lifecycle Management, in form of a prototypical web application with chosen collaborative features as well as tools that accelerate workflows and secure a successful, qualitative API development. Moreover, the whole thesis is conducted in cooperation with an industry partner from the financial sector in Germany.

This thesis applies the design science framework from Hevner et al. 2004. The design artifact of this thesis is the prototypical implementation of the Collaborative API Lifecycle Management. This includes the collection of requirements from literature and expert interviews to form a minor artifact which is a conceptual model for the API Lifecycle. The design artifact is evaluated in form of a case study with expert interviews. The results show that the prototype is a viable solution. The proposed acceleration approach helps to increase the speed of the whole lifecycle process. The chosen collaboration features are perceived positively. However, collaboration generally needs active commitment otherwise the features have no use.

Keywords: API, API Economy, API Management, Full Lifecycle API Management, API Lifecycle Management, API First, API Governance, Collaboration Engineering, Co-Creation

Contents

List of Figures

List of Tables

List of Abbreviations

ADR	Architectural Decision Record
API	Application Programming Interface
BFF	Backend for Frontend
BPM	Business Process Management
BPMN	Business Process Model and Notation
CALM	Collaborative API Lifecycle Management
CD	Continuous Delivery
CI	Continuous Integration
CMMN	Case Management Model and Notation
DSR	Design Science Research
DX	Developer Experience
EA	Enterprise Architect
HTTP	Hypertext Transfer Protocol
ITSM	IT Servive Management
MEAN	MongoDB ExpressJS Angular NodeJS
PDSA	Plan Do Study Act
PoC	Proof of Concept

RBAC	Role Based Access Control
REST	Representational State Transfer
SCM	Source Code Management
SDK	Software Development Kit
SDLC	Software Development Lifecycle
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SUS	System Usability Scale
UX	User Experience

1. Introduction

1.1. Motivation

The idea behind Application Programming Interfaces (API) has been in science and industry for many decades (Collins and Sisk 2015). The dissertation of Fielding 2000 marked the beginning of today's web APIs. Fielding 2000's contribution set the fundamentals and since then, numerous web APIs have been developed over the last decades. Today, APIs and web APIs are treated as synonyms (De 2017). In fact, the last decade reveals that firstly the number of APIs and secondly the usage of APIs have grown exponentially (Programmable Web 2018). Even though these statistical numbers are only visible for public APIs, experts assume that the number of private APIs is much larger than the publicly accessible ones (Jacobson et al. 2011; De 2017; Postman 2017).

Numerous businesses are slowly turning into a digital business and APIs already became the basis for various companies in areas like Cloud Computing, Internet of Things (IoT) and Big Data (W. Tan et al. 2016; CA Technologies 2015). With the new introduction of the PSD2 law, Banks even had to provide APIs to allow other companies access to their encapsulated data and services (Nordic Apis 2016; Brodsky and Oakes 2017). In fact, the financial technology industry experienced also a high usage of APIs recently. Moreover, companies discovered the emerging trend of the API Economy where companies treat their APIs as a product and monetize them by giving users or partners access to their back-end functionalities as well as data properties via the APIs. By making APIs accessible to external or partner consumers, companies are able to reach new markets, enable their business strategy and drive the creation of new innovative solutions (Iyengar et al. 2017; Kepes 2014; Holly et al. 2014).

To constantly profit from the API economy, companies need to pay high attention to a good API Management. API Management is *„a crucial capability to navigate the digital age"* (Iyengar et al. 2017). Part of the API Management is the API Lifecycle Management. Generally, API Management sets the focus to the service side of an API, while the API Lifecycle Management handles the product or rather technical side. However, traditional API Management systems seem to treat API Lifecycle Management as a minor role. To

have a successful API strategy in the API economy, it is important to give API Lifecycle Management the same attention as to API Management. Therefore, a trend towards Full Lifecycle API Management has been developed recently.

1.2. Problem Statement

Although there is a high potential on the API market, APIs still suffer from a variety of issues. Firstly, APIs are often not well designed that lead to bad user experience (UX) or rather developer experience (DX) (Bermbach and Wittern 2016; Smith 2018; Earls 2013; Myers and Stylos 2016; Bermbach and Wittern 2016). Developers, using the APIs, wish to have more documentation that is properly described to make the usage of the APIs easier (Postman 2017).

Secondly, APIs need to be more consumer-driven. Many companies do not collaborate with external users and provide a rather provider-driven API that does not fulfill the consumer's needs (Smith 2018). Generally, collaboration is inherently needed, in order to scale an API program (Whitehead 2007; Vasudevan 2017). *„The lack of an easy way for stakeholders to quickly access the right APIs at the right time, while collaborating without confusion on new API definitions, [...] [affects the] overall API delivery speed"* (Vasudevan 2017).

Lastly, when developing for instance a mobile app, there are dependencies between front-end and back-end developers teams which lead to longer development cycles for the whole app. While the back-end team can implement the logic of a needed API, the front-end team needs to wait until the API is ready and provided by the back-end counterpart (Rivero et al. 2013).

1.3. Research Questions

In the following section, a set of research questions (RQ) are presented which are derived from the previously described problem statement and guide this thesis throughout the whole research.

RQ1: How could a holistic approach for an API Lifecycle, including phases, activities, artifacts and roles, look like that is driven by the collaboration of participating stakeholders?

The aim of the first research question is to create a holistic lifecycle for APIs that builds the target lifecycle model for the industry partner. The designed API Lifecycle should address the challenges from literature, but also challenges which the industry partner has with its current process for APIs. Moreover, the lifecycle should have a strong emphasis on collaboration and agility. Before forming the conceptual model, requirements of the API Lifecycle are collected by literature review and expert interviews. Subsequently, phases, activities, artifacts and roles of the API Lifecycle model are collected in the same way as the requirements. Based on the previously gained knowledge, the API Lifecycle model is finally conceived as artifact of this research question.

RQ2: How can tools and collaborative features be used to support the API Lifecycle Management?

To answer this question, a web application prototype is designed and implemented to assist the different stakeholders during the first initial stages of the API Lifecycle, which was conceptualized in RQ1. For that, the system design of the prototype is built and presented. In addition, suitable collaboration features are chosen from a pool of possible collaboration tools that are categorized by (Leimeister 2014) along space and time. The chosen features are then included into the prototype to drive collaboration among the participating stakeholders throughout the whole API Lifecycle. Besides the collaboration features, supplementary features, accelerating the API Lifecycle Management, are introduced.

RQ3: What are the users' experiences of the designed web application prototype solution?

The prototype is evaluated in form of a case study including multiple expert interviews. For that, two scenarios with a sequence of activities are prepared. The first scenario covers typical activities of the API consumer. This includes for example the creation of new API proposals or searching for existing ones. Similarly, the second scenario covers activities of the API provider side. Particularly, enterprise architects are the main focus in this scenario. A typical assignment of an enterprise architect would be to approve or reject a proposal which is based on the provided proposal details and pre-approvals from key decision makers for project funding and governance conformity. After each scenario, the system usability scale (SUS) evaluation method is applied. Lastly, the gained key findings and limitations of the current web application prototype are summarized.

1.4. Research Approach

The whole research approach of this thesis is based on the design science framework, defined by (Hevner et al. 2004). Figure 1.1 provides an overview about the research framework which was adapted to this thesis' context. The framework provides guidelines that this thesis systematically follows, in order to provide a solution for Collaborative API Lifecycle Management. The goal of this framework is to provide an IT artifact that is driven by a business need and fundamentally built on a rigorous knowledge base. As illustrated in the design science model, the framework consists of three main parts which are the environment, the information systems (IS) research and the knowledge base.

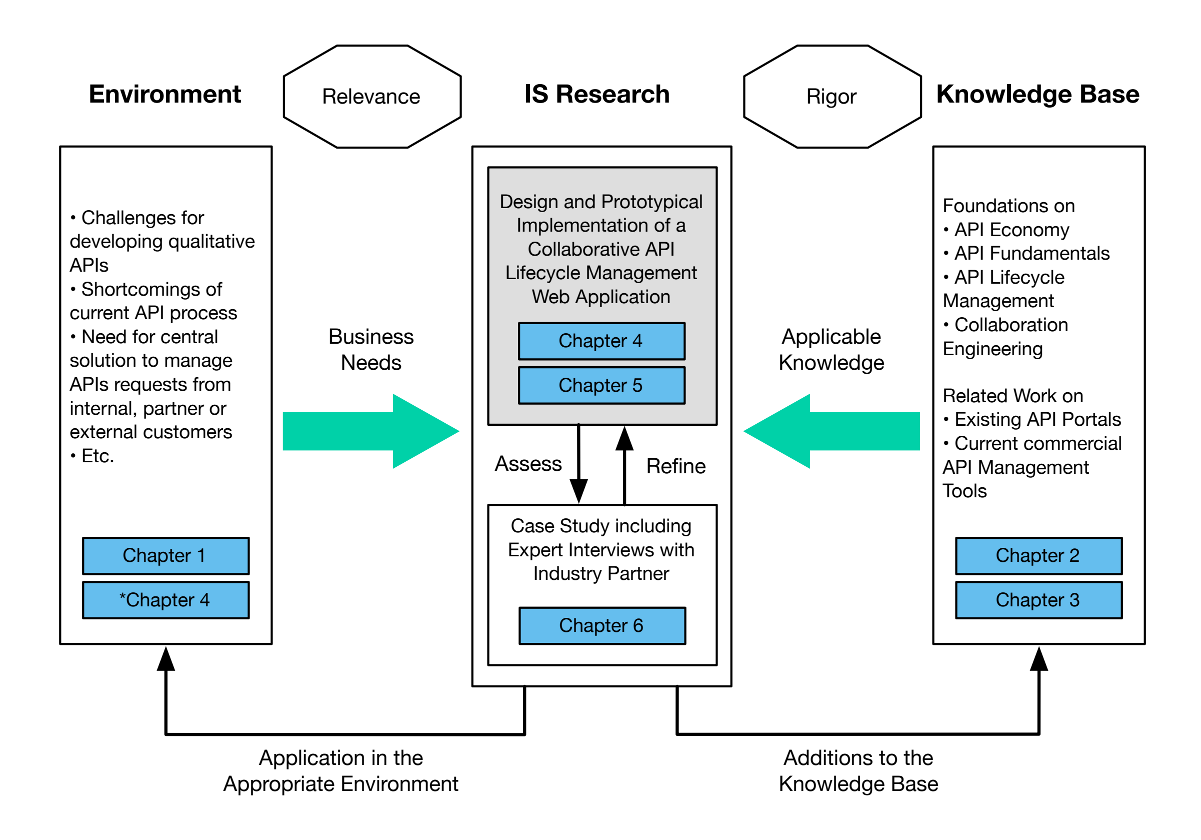


Figure 1.1.: Information Systems Research Framework with Thesis Contribution. *Adapted from (Hevner et al. 2004).*

The environment describes the problem space which is composed of „people, (business) organizations, and their existing or planned technologies" (Hevner et al. 2004). The environment defines a business need that people in an organization perceive. People consider this business need as crucial to perform certain tasks and reach defined goals effectively as well as efficiently. It influences the IS research, in order to achieve relevance

between the IS research and the given environment. In this thesis, chapter 1 describes the general problem space and subsection 4.1.1 in chapter 4 adds further challenges from the industry partner to the environment.

The knowledge base includes foundations, methodologies and related research. It provides the fundamental knowledge to build the IT artifact and fulfill the business need in the given environment. Rigor is achieved by applying the knowledge appropriately. The knowledge base for this thesis is covered by chapter 2 and 3.

The IS research is divided into two phases which are complementary to each other. The first phase is the „Develop/Build“ phase and the second one is the „Justify/Evaluate“ phase. The heart of the whole design science framework is to conceptualize and develop an IT artifact via the „Develop/Build“ phase which is afterwards evaluated via the „Justify/Evaluate“ phase by applying the IT artifact in the given environment, in order to satisfy the business need. The feedback gained from the evaluation is used on the one hand to improve the IT artifact, but also on the other hand to reshape the knowledge base. This assessment and refinement cycle is ideally done in several iterations. Due to the time constraint of six months for this master's thesis, the assessment and refinement cycle in this thesis is only performed once. In this thesis, chapter 4 and 5 provide the IT artifact in the „Develop/Build“ phase. Chapter 6 performs the evaluation of the IT artifact in the „Justify/Evaluate“ phase and presents the results.

As recommended by (Hevner et al. 2004), this thesis follows the seven guidelines for conducting effective design-science research:

Guideline 1: Design as an Artifact In this thesis, a collaborative approach for API Lifecycle Management is designed and developed. Specifically, this Collaborative API Lifecycle Management addresses the business need in the industry partner's business environment. As outlined in chapter 5, a corresponding prototype with its implementation details is introduced. This prototype builds the main design artifact in the sense of the design science research. It is considered as viable artifact in form of an *instantiation* and serves moreover as proof of concept.

Guideline 2: Problem Relevance The environment in this thesis includes general challenges from literature review about providing qualitative APIs which fit to the user's needs. There is a need to collaborate between API consumer and provider, in order to form more consumer-driven APIs. The details to the problems are described in section 1.2. Further challenges are collected in form of expert interviews with the industry partner. The results are summarized in subsection 4.1.1.

Guideline 3: Design Evaluation Chapter 6 describes the evaluation approach as well as the results. In general, the evaluation approach consists of a case study, including expert interviews. The validation of the IT artifact is conducted only in one iteration, due to the time constraint of six months for this thesis. The evaluation uses the SUS evaluation method, developed by (Brooke 1996). Particularly, SUS focuses on the usability of the design artifact which inherently includes properties like effectiveness, efficiency and satisfaction. As elaborated in section 4.2, a scenario is created for both API consumer and API provider which is afterwards evaluated by the interviewee with the standardized SUS questionnaire. After conducting the scenarios, further open questions are asked to gain general feedback and pinpoint limitations of the current solution approach.

Guideline 4: Research Contributions The first major contribution of this thesis consists of the conceptual model and the requirements for a collaborative approach for API Lifecycle Management which are amplified in section 4.1 and 4.2. Specifically, the conceptual model is derived from the foundations, related research and expert interviews to meet the requirements. The conceptual model is adapted to the industry partner's needs and is not to be considered as a generic model, applicable to different companies. Another major contribution is the prototypical implementation to support the Collaborative API Lifecycle Management. This prototype inherently serves a proof of concept, while its evaluation results gives insights into the usability of the current prototype and the utility of the whole designed solution approach.

Guideline 5: Research Rigor The foundations and related research are outlined in chapter 2 and 3. Built on the gained knowledge base and further researched requirements, a conceptual model for Collaborative API Lifecycle Management is derived. Furthermore, collaborative features are analyzed and appropriate ones are selected for the respective prototype. The evaluation results of the IT artifact highlight possible enhancement aspects not only in the usability of the prototype's user interface (UI), but also in the foundations and the conceptional design. As a result, the gained evaluation feedback can be used for future research.

Guideline 6: Design as a Search Process As already mentioned in this section, there is only one iteration of the assessment and refinement cycle between building and evaluating the IT artifact. Nevertheless, the results from the evaluation help to redesign both the conceptual model and the prototypical implementation. Moreover, it highlights limitations of the current solution and points out further future research. A summary of all key findings are concluded in the last chapter of this thesis (cf. chapter 7).

Guideline 7: Communication of Research The general approach and the key findings of this master's thesis are presented in two presentations, during the graduation seminar of the sebis chair from Technical University of Munich (TUM), Germany.

1.5. Outline of this Thesis

As illustrated in Figure 1.1, each step of the design science framework, defined by Hevner et al. 2004, is described in at least one chapter of this thesis. Additionally, the following section provides a short insight into each chapter:

Chapter 1: Introduction motivates the thesis and describes the addressed problem space. Furthermore, the research questions and the research approach are elaborated.

Chapter 2: Foundations provides a theoretical basis for this thesis. This chapter covers important terms and concepts, including API Fundamentals, API Economy, API Lifecycle Management, including API Management, and Collaboration Engineering.

Chapter 3: Related Work gives an overview about related research in the area of API Management and API Lifecycle Management. Since the current academic literature in this research area is sparse, mainly related research from industry is presented. This includes existing API Portals from other companies and other commercial API Management tools.

Chapter 4: Conceptual Design amplifies different aspects of the design of a Collaborative API Lifecycle Management prototype. This chapter covers the requirements of a conceptual model for Collaborative API Lifecycle Management and also presents the result of the designed model. The requirements contain additional challenges from the industry partner, from which most of the challenges are addressed by the conceptual API Lifecyle Management model. Chosen collaboration features, which are included into the prototype, are introduced. Moreover, the system design of the prototype is elaborated which includes an use case diagram, a role-based access control (RBAC) model, a data model and the overall activity flow for the first initial stages as a state machine diagram.

Chapter 5: Prototypical Implementation presents the realized prototype. This chapter gives an overview about the used technologies and lays out possible alternatives. The overall architecture is presented, followed by a description of each single component of the front-end as well as back-end.

Chapter 6: Evaluation provides the evaluation goal, the evaluation approach and the feedback gained by expert interviews. Moreover, possible enhancements for the prototype are proposed.

Chapter 7: Conclusion summarizes the results of the research questions and concentrates on the key findings achieved in this thesis. Besides, limitations and future work are highlighted.

2. Foundations

The following chapter provides the knowledge base in the sense of Hevner et al. 2004. It presents an overview of important terms, concepts, and methodologies used in this thesis. Based on the knowledge base, the design artifact, a prototypical solution for Collaborative API Lifecycle Management, is designed and implemented. To begin with, the API fundamentals are presented which builds the essential basis to understand subsequent concepts. After that, an insight is given into API Economy and API Lifecycle Management. Finally, the concepts of Collaboration Engineering are introduced.

2.1. API Fundamentals

An API and a web service are closely related to each other. In fact, people still mix up the two terms and use them interchangeably. Even though both may seem to follow the same purpose, yet both still show different characteristics. To distinct both terms, they are explained in the following.

In general, API is the abbreviation for *application programming interface*. An API can be technically described as *„a way for two computer applications to talk to each other over a network [. . .] using a common language that they both understand“* (Jacobson et al. 2011). The communication over a network is performed without any user interaction (De 2017). The network does not have to be via the web. However, if the web is used as the network, then the API is also called *Web API* (De 2017). In this thesis, the terms *API* and *Web API* are used as synonyms. The purpose of an API is to expose data or functions to be used in other applications (De 2017; Masse 2012). A typical example is the Google Maps API. The Google Maps API provides geographical data that can be accessed by developers via an API and used inside the developer’s mobile app. Another point is that APIs build a contract (Fremantle et al. 2015; Jacobson et al. 2011; Souza et al. 2004). Ideally, this means that API are reliable which increases usage. In addition, good APIs are documented, consistent, and predictable (Jacobson et al. 2011).

A web service is *„a software system designed to support interoperable machine-to-machine interaction over a network"* (W3C 2018). *„A Web API is the face of a web service, directly listening and responding to client requests."* (Masse 2012). De 2017 even mentions that a web API is a subset of web services. Web services are typically based on the communication protocol SOAP, while web APIs are based on REST (De 2017; W3C 2018; Spichale 2017). SOAP makes data available as services while REST makes data available as resources (Patni 2017; Vester 2017). There are also other protocols for APIs which include GraphQL, RPC, etc. All of them have advantages and disadvantages. Nevertheless, REST is currently the most used one in the industry.

2.2. API Economy

After having the basic idea what an API does, the following section gives a brief insight into the API economy which has gained high attention by many companies over the last years. Firstly, the term itself is introduced. Secondly, the API value chain is presented and lastly the API types are amplified.

2.2.1. Key Terms

Overall, there is no universal definition for the term *API Economy* in the current literature. The general idea of the API economy can be described as *„the commercial exchange of business functions, capabilities, or competencies as services using web APIs"* (Holly et al. 2014). The API has shifted from a mere *„development technique to a business model driver"* (Collins and Sisk 2015). APIs can be considered as a product and offered to other users who makes use of APIs to create new business opportunities. Moreover, companies can monetize the locked up data and services via APIs which at the end provides the company a new revenue stream (Vukovic et al. 2016). Typical business models include subscription, license, freemium, or pay-as-you-go (Vukovic et al. 2016; Jacobson et al. 2011).

2.2.2. API Value Chain

The API economy creates value for a number of different actors along the API value chain. Figure 2.1 shows how an asset can be transformed into a value for the end users.

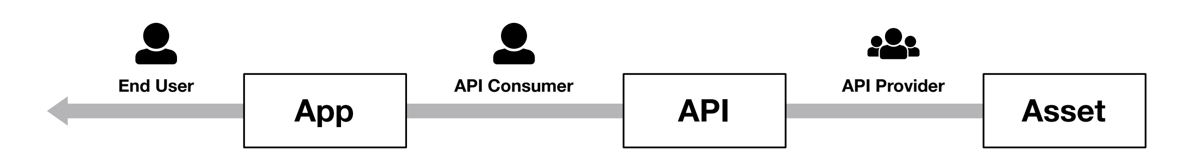


Figure 2.1.: The API value chain (*adapted from (De 2017)*)

The API value chain starts with a business asset which can be information, a product or a service (Jacobson et al. 2011). The API provider exposes this asset via an API and offers it to API consumers. The API consumer, who equals to a developer, uses this API to build for instance a new mobile app. This mobile app is finally used by the end user. Value is provided to *„the owner of the assets, the publisher of the API, or the developers who created the applications"* (Jacobson et al. 2011).

All in all, all key stakeholders in the API value chain need to be fully involved and committed to create strong links and have a successful API strategy (Jacobson et al. 2011; De 2017). API strategies often fail due to weak links in the API value chain (Jacobson et al. 2011).

2.2.3. API Types

There are several ways to divide APIs into categories (Brodsky and Oakes 2017; De 2017; Jacobson et al. 2011; Morgan et al. 2016). The majority of authors suggests that APIs can be divided into two big groups which are public and private APIs. The private APIs can be further divided into internal and partner APIs. Overall, an API may fit into several categories.

Internal APIs are used merely within the company and are not accessible to the public. They help to *„improve organizational agility, efficiency, and effectiveness"* (Morgan et al. 2016).

Partner APIs enable *„highly customized integration with selected business partners, customers, and other stakeholders, typically within the context of specific business processes and relationships"* (Morgan et al. 2016).

Public APIs are accessible by anyone and is used to *„expand market reach through openness and innovation"* (Morgan et al. 2016). They help to create new business ideas and reduce development costs (De 2017).

At first glance, public APIs seem to contribute most to the API economy. In fact, the private APIs are the actual driving force in the API economy (De 2017; Jacobson et al. 2011). Most public APIs started as a private one first and later became open to all developers.

2.3. API Lifecycle Management

To constantly profit from the API economy, a company requires a well planned API Management which also includes a good API Lifecycle Management. This coming section, starts with introducing the basic terms like API Management and API Lifecycle Management. Subsequently, an insight into the API First design is provided. Finally, a comparison of similar lifecycle approaches is given.

2.3.1. Key Terms

Currently only a few academic research about API Management has been published. Hence, there is no universal definition for the term *API Management*. Spichale 2017 describes API Management as all technical, economic and strategic aspects that are needed to operate APIs for a company or a service. API Management is often represented as a platform that *„helps an organization publish APIs to internal, partner, and external developers to unlock the unique potential of their assets"* (De 2017). API Management offers a variety of capabilities (Patni 2017; Fremantle et al. 2015; Spichale 2017), but De 2017 reduces the capabilities to four groups:

- Developer Enablement for APIs
- Secure, Reliable and Flexible Communications
- API Lifecycle Management
- API Auditing, Logging and Analytics

These capabilities are offered in an API Management platform as three major types of services (De 2017; Stafford 2018):

1. API Gateway service
2. Analytics service

3. Developer portal

The API gateway service allows to create and manage APIs. They add functions to APIs which include *„security, traffic management, interface translation, orchestration, and routing capabilities"* (De 2017). The analytics service monitors traffic, operational metrics, API performance, and developer engagement metrics (De 2017). The developer portal is the single entrance point for developers. Functions include *„app registration and onboarding, API documentation, community management, and API monetization"* (De 2017).

The previously described characteristics apply only to traditional API Management systems. Stafford 2018 mentions that the traditional API Management shifts towards Full Lifecycle API Management. In general, this means that traditional tools gave lower attention to the API lifecycle. The API Lifecycle kept playing a minor role and needs to get higher attention now. Researchers discovered that companies do not have a formal lifecycle management. Therefore, it is necessary to shift to Full Lifecycle API Management which allows to control the API workflow. The major services offered by the traditional API Management systems is extended and results into the following (Stafford 2018):

1. API Design
2. API Implementation
3. API Testing
4. API Gateway service
5. Analytics service
6. Developer portal

Every company has its own lifecycle for APIs. Hence there is also no universal definition for the term *API Lifecycle Management*. Generally, API Lifecycle Management *„provides the capability to control how an API is developed and released to consumers"* (De 2017). Basic functions include API Creation, API publication, change notification, version management, issue management, API deprecation and retirement (De 2017; Patni 2017).

2.3.2. API First Design

Companies often start with creating the product itself like for instance a mobile app or web application. They treat APIs as an integration tool to glue third party systems and services together which are then used to support the back-end functionality of the actual product. This results into several customized, artificial APIs that are not properly built, tested and do not contribute to reusability. However, the goal in the API economy is to create APIs which developers can benefit from and which can be reused for more than one application.

An alternative approach that has been present in the industry for some years is API First. API First *„is a strategy in which the first order of business is to develop an API that puts your target developer's interests first and then build the product on top of it [...]"* (Levin 2016). Common practice is to create an API specification as a Swagger file, the OpenAPI Standard, and define the details of the endpoints, including request and response formats. Using the API specification as basis, the actual product can be implemented afterwards.

Figure 2.2 shows the order of steps in the Code First approach. The Code First approach starts with developing the back-end service logic which is followed by the API creation. After the API has been implemented, the front-end team can start to implement the actual product's visual representation. It can be clearly seen that there is a dependency between the development teams. There is also no space for parallelism. This way of development is also called synchronous development (Levin 2016).

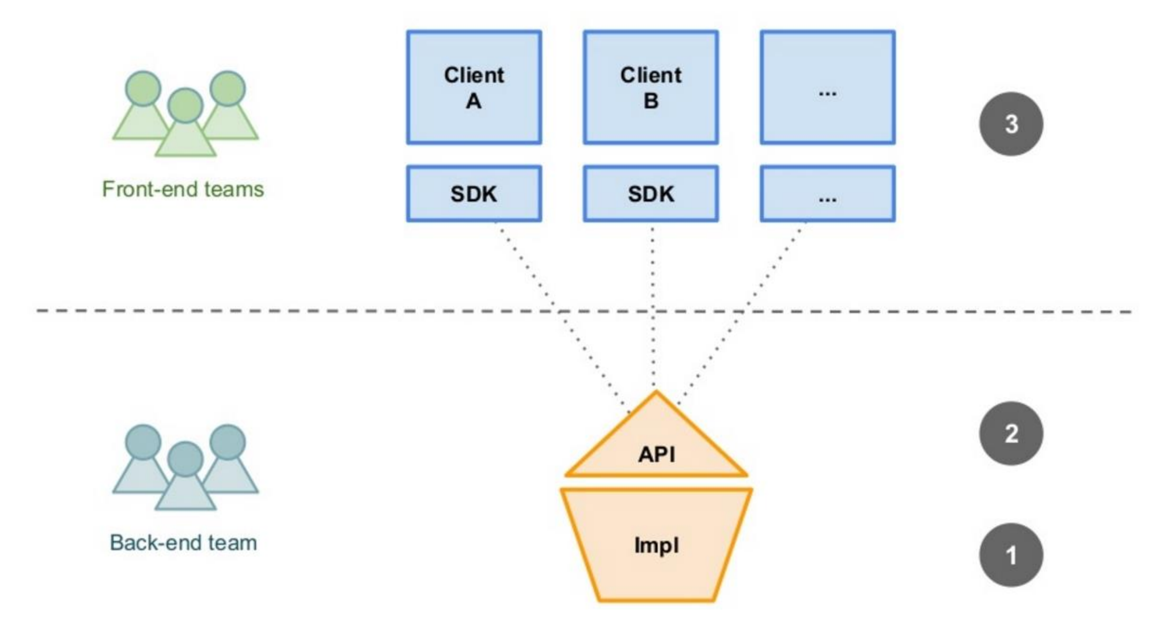


Figure 2.2.: Code First Approach (Levin 2016)

The Code First approach leads to longer development cycles, whereas the API First approach allows for parallelism to develop the parts of the product faster. As shown in Figure 2.3, the parallelism is achieved by mocking the API specification to separate front-end and back-end teams (Rivero et al. 2013). After that, the back-end functionality and the front-end of the product can be implemented at the same time. Both back-end and front-end team are collaborating based on the API mock.

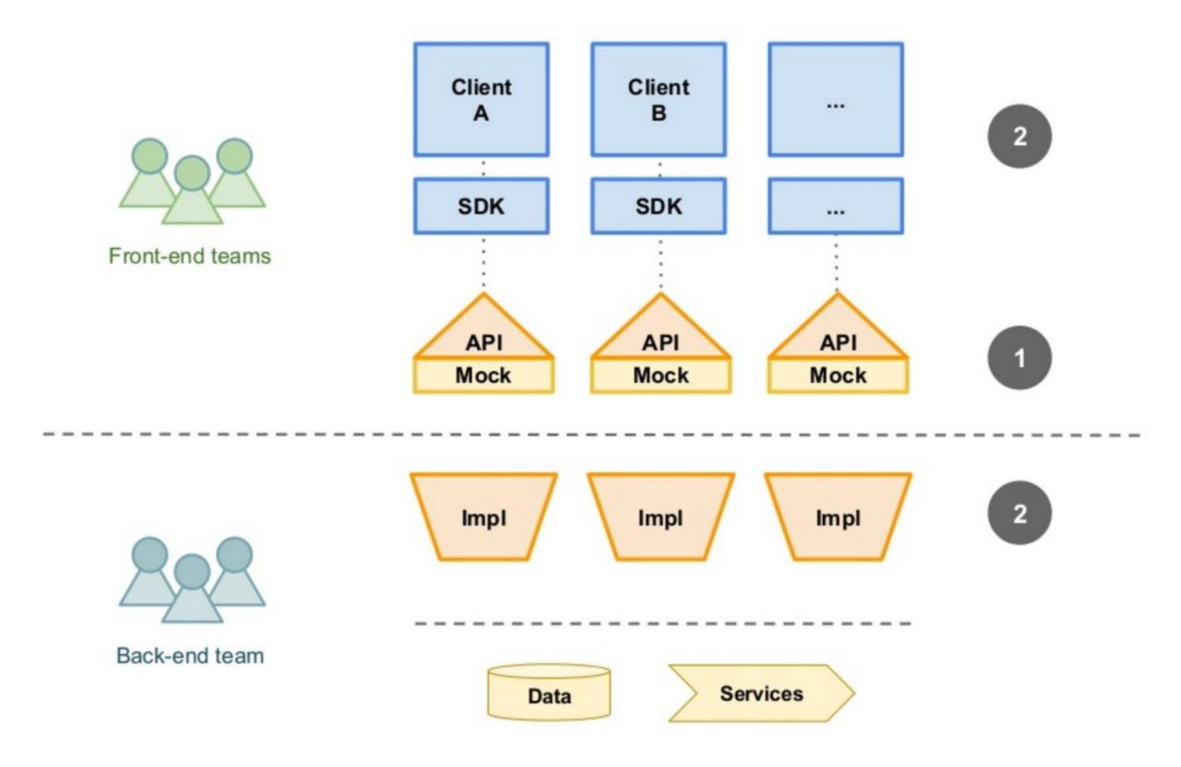


Figure 2.3.: API First Approach (Levin 2016)

2.3.3. Comparison of Similar Lifecycle Approaches

By comparing existing API Lifecycles from literature and current industry leaders (Malinverno and O'Neill 2018; Malinverno and O'Neill 2016), it can be seen that there is no standardized one and that every company has its own lifecycle model for APIs (Apigee 2016a; Ravichandran et al. 2016; Patni 2017; CA Technologies 2015; Vester 2017). There are API Lifecycle models which only consider the API provider angle and there are others which have a combined API consumer and provider perspective.

Because of the characteristics of an API, the API Lifecycle might have similarities with the IT service management (ITSM) lifecycle and the software development lifecycle (SDLC). Both lifecycles are further analyzed. Table 2.1 shows the comparison of the SDLC and the ITSM. It can be seen that the SDLC focuses more on the product, while the ITSM

concentrates on the service. Similar parts of both lifecycles are also visible in an API Lifecycle. Hence, an API Lifecycle has both a product and a service focus. At first, an API Lifecycle includes the development of the actual product which is an API. Furthermore, the API Lifecycle provides additional services like operating the API to give consumers access to the product. Compared to the SDLC, the API Lifecycle creates a product, but it stays with the API provider. Therefore, the API Lifecycle needs to consider end-to-end business services.

Table 2.1.: The comparison of SDLC and ITSM (*adapted from (Pollard et al. 2009)*)

Product Focus (SDLC)	**Service Focus (ITSM)**
Planning	
Negotiate scope based on function	Negotiate scope based on end-to-end business process
Internally (IT) focused	Customer focused
IT jargon	Business jargon
Requirements Modeling	
„Over the wall" mindset	Stakeholder involvement
Technology insights	Business metrics
Automate function once and move on	Automate service once reuse service in different ways
Focus on inputs and outputs	Focus on business needs and process
Design	
Capture logic of business function	Model business rules and external relationships
Focus on IT artifact	Focus on end-to-end business services
Construction	
Create a software product	Increase focus on value-added portions of applications
Buy, build or lease	Buy, build, lease and INTEGRATE
Deployment	
Technology driven	Minimum impact on business services
Test technology	Test service environment
Train on technology	Train in business service/process
Support	
Maintain hardware/software/networks	Continual service improvement

The comparison of further lifecycles from the area of SOA governance (Schepers et al. 2008) and service management (Fischbach et al. 2013; Kohlborn et al. 2009) with the API Lifecycle leads to the conclusion that all lifecycles can be reduced to the *Deming Cycle* which is also called the Plan-Do-Study-Act (PDSA) cycle (The W. Edwards Deming Institute 2018). The

stages of all the lifecycles can be allocated to one of the PDSA cycle. The only difference is that an API Lifecycle includes stages with API-specific activities. The naming of the stages are similar to other lifecycles.

2.4. Collaboration Engineering

Software Engineering normally includes a variety of people with different roles. All of them have different tasks and responsibilities, but all achieve one common goal together. Hence, collaboration is inherently part of software engineering. The following section gives an overview about the concepts of collaboration engineering. At first, important terms related to collaboration are introduced which is followed by a classification of collaboration tools.

2.4.1. Key Terms

Collaboration and cooperation are two common words that are used by people interchangeably. In fact, both terms are not synonyms. Collaboration reflects an umbrella term and is further divided into communication, coordination and cooperation. Leimeister 2014 presents the relationship of all four terms and describes it as the *„four Cs of collaboration"*, shown in Figure 2.4.

The lower half shows the triangular relationship between communication, coordination and cooperation. This constellation is often referred to as the *„3C collaboration model"* and builds the basis for collaboration. The first basic term is **Communication** and can be defined as *„the interrelated behavior of two or more people and their interaction with the goal of transmitting information and understanding the content."* (Leimeister 2014).

The next basic term related to collaboration is coordination. **Coordination** can be described as *„the matching of decentralized actions and decisions of interdependent organizational units on the basis of suitable communication processes with regard to the optimal fulfillment of the goals. [...] Coordinated systems can work in parallel and are uninfluenced by each other"* (Leimeister 2014). Communication is used for coordination. All coordinated actions are independent from each other and no common medium is needed.

The last basic term is cooperation. **Cooperation** can be defined as *„the activity of two or more individuals, which is consciously planned and coordinated with one another to ensure the achievement of the goals of each individual involved to the same extent"* (Leimeister 2014). Compared to coordination, cooperation needs a common medium to work.

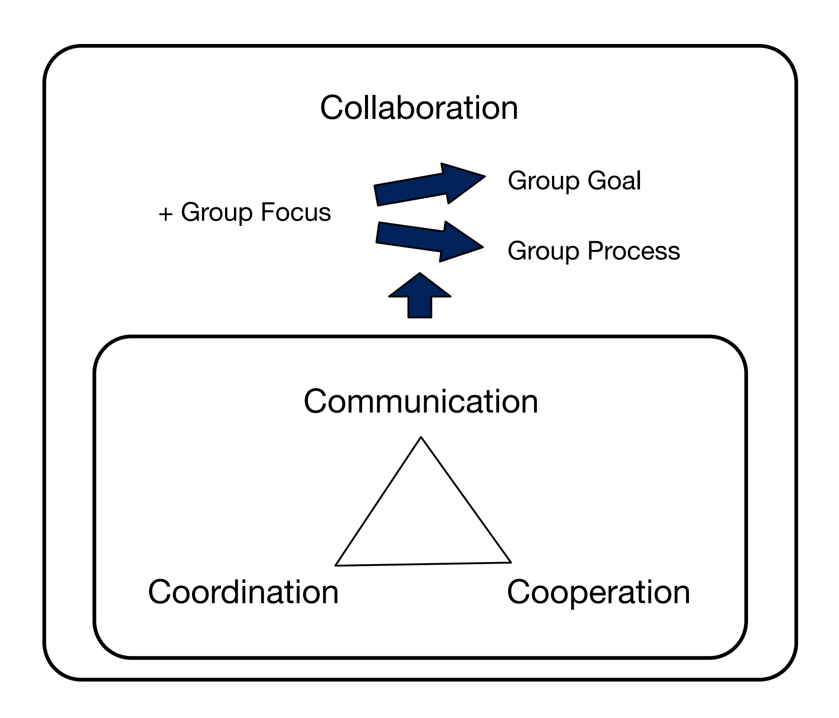


Figure 2.4.: The four Cs of collaboration (*adapted from (Leimeister 2014)*)

All three terms are closely related to collaboration. As already mentioned, cooperation and coordination are often used as synonyms. However, as depicted in the *„4Cs of Collaboration“* model, it can be seen that collaboration pays special attention to the group focus. Communication, coordination and cooperation concentrate on interactions of single actors, whereas collaboration focuses on fulfilling the group goal. To achieve the group goal, the group process needs to be considered. In general, **Collaboration** can be described as two or more individuals working on a common medium to fulfill a common goal, and in order to reach this goal communication, coordination and cooperation of the involved actors are necessary (Leimeister 2014).

2.4.2. Tool Classification

To simplify collaboration, there are a variety of collaboration tools that can be used online and offline. A collaboration tool depends on the interacting people if they collaborate at the same place or different places and also if they collaborate at the same time or different times. The combination creates a space-time-matrix with four classification dimensions which is depicted in Table 2.2.

The first dimension requires that collaboration happens at the same time and at the same place. This means that a group of people could be located in the same meeting room. All

used tools in that room are then treated also as collaboration tools. Further suitable tools are for instance presentation tools like a flip chart or shared editor tools like tabletops.

Table 2.2.: Classification of collaboration tools along space and time (*adapted from (Leimeister 2014)*)

	Same Time (Synchronous)	Different Time (Asynchronous)
Same Place	**1st Dimension** Brainstorming Tool Whiteboard Flip Chart Voting Tool Overhead Projector Data Projector Tabletop Pin Board	**3rd Dimension** Message Board Common Work Room Answering Machine Adhesive Labels
Different Place	**2nd Dimension** Instant Messaging System File Transfer Tool Phone VoIP System Desktop/Application Sharing Tool	**4th Dimension** E-mail Newsgroup Bulletin Board Web Blog Wiki Video Streaming Platform Shared File Repository Social Tagging Voting Tool

The second dimension covers all tools that requires people to collaborate at the same time, but at different places. A typical example is when people work in different cities, but need to collaborate together to a certain time. Common tools include conversation tools like instant messaging or desktop sharing tools.

The third dimension includes tools that are helpful for people working at different times, but at the same place. Generally, this dimension covers only a few tools and people often concentrate more on the other dimensions (Leimeister 2014). Suitable tools cover for instance a message board or adhesive labels like post-it notes.

Finally, the last fourth dimension covers tools for collaborating at different times and different places. Those tools give the most flexibility and includes the most collaboration tools, compared to the other dimensions. This dimension includes both a variety of online as well as offline tools. Common tools for this dimension are conversation tools like email, bulletin boards or weblog. Further tools include presentation tools and voting tools.

3. Related Work

The current academic literature about API Management and API Lifecycle Management is sparse at this point in time. However, the industry has seen the potential of APIs early and is actively participating on the API market. A decent number of white papers has been published by leading industry players about API Management. Since the goal of the design artifact is to support a full lifecycle API Management, the prototypical implementation of the design artifact can be treated as an API portal that brings API consumer and provider to one central place. In the following, existing API portals and a few commercial tools in the area of API Management and API Lifecycle Management are presented as related research.

Daimler 2018 owns an API portal that holds a few number of APIs about cars. The APIs can be tested by subscribing them and getting an access token. On the portal are different ways to contact the provider for example a form and an email address. The form can be used to request changes or a new API. A documentation and a pricing list is also displayed on the portal.

Nasa 2018 provides an API portal about space related topics. The portal shows how to subscribe to existing APIs. The subscription requires the API consumer to fill out a contact form. Generally, the API portal offers only Open APIs which are all publicly accessible to all API consumers. There is no pricing for the usage. In fact, developers can contribute to the API portal by offering own APIs to the company.

Google 2018 offers *Apigee* as an API Management tool that claims to support the Full Lifecycle API Management. By trying out the API Management tool, it can be seen that the company offers a variety of typical API Management characteristics like developer enablement, analytics and security features. However, it seems that *Apigee* sets a focus more on the service side which means that API are published and its usage is tracked for optimization. There is neither a way for automation of the development environment nor any collaboration support.

Smartbear 2018 offers also API Management tools. Most of the tools focus on the automation of the development environment which is also based on an API Specification.

Collaboration along the development is also supported. However, the company focus only on the technical side. The full lifecycle for APIs is not considered, only a part is covered.

All in all, the presented API portals and API Management tools cover only parts of a Full Lifecycle API Management. Most of the tools do not support collaboration along the whole lifecycle. There are tools that automate the development environment and provide some collaboration features like a form and email for contacting. Nevertheless, a complete end-to-end service seems not to be given by any existing company at the current point in time. In addition, not all the existing API Lifecycle Management tools have a product and service focus at the same time.

4. Conceptual Design

After acquiring the foundations and related work for this thesis, the subsequent step in the DSR is to build the design artifact which uses the business need and the acquired knowledge base as an input. Before creating the actual design-artifact, a conceptual model and the system design of the IT artifact need to be prepared. In order to form the conceptual model for API Lifecycle Management, requirements from literature as well as expert interviews are collected. In combination with further expert interviews and the research foundations, the actual lifecycle model is designed. In addition, the chosen collaboration tools are presented and the IT artifact's system design are elaborated.

4.1. API Lifecycle Requirements

To form the fundament of the conceptual model for API Lifecycle Management, requirements are collected and presented in this coming section. At first, challenges from the industry partner are elaborated which are then followed by introducing success factors from similar or existing API Lifecycle approaches in literature.

4.1.1. Challenges from the Industry Partner

By interviewing two enterprise architects from the industry partner, a number of major challenges could be identified that currently hinder a successful service and API delivery to internal and external customers. The major challenges are summarized in Table 4.1. The following section briefly describes each challenge individually.

The first challenge *C1* addresses the problem about status updates of projects. A project normally consists of a lot of different roles. Each role is responsible for a part of the project. Due to the high number of interacting people, collaboration is inherently needed. Therefore, it is crucial that all people need to be informed about the current project status or at least, the project members are able to get the missing information instantly by themselves. However, the industry partner mentions that it is currently difficult to align

Table 4.1.: The industry partner's major challenges with services and APIs

	Major Challenge
C1	Difficulties to align all project participants on the same status of an API
C2	Unnecessary longer and indirect communication ways
C3	Neither traceability nor transparency with classical communication ways like phone or email
C4	Break in collaboration among project members
C5	External API customers have no contact point to request a new API
C6	High manual paperwork for use of non-optimized API process
C7	Existing tools are cumbersome and difficult to use
C8	Customer not involved into the whole process and cannot provide early feedback

all project members. The only way to be informed is to personally attend meetings or wait until someone creates a report. Generally, a person is not able to get the status instantly.

The next challenge *C2* deals with ineffective communication ways. Currently, a developer in a project is not able to contact the customer directly. The industry partner states that the company is divided into domains. To transmit messages from the developer to the customer, a domain architect or a similar role need to be used as a „broker". This means customers generally do not exchange communication with developers directly.

Challenge *C3* describes the disadvantages of traditional communication ways. In general, the industry partner still uses traditional communication ways like phone or email. Those communication ways are still needed and cannot be replaced by anything else yet. Nevertheless, without additional communication ways, it is difficult to trace changes and to match the information to the right projects. The industry partner mentions that transparency is generally missing.

The challenge *C4* identifies that sometimes there are breaks in the collaboration process. One of the interviewed enterprise architect adds that at first, a service or API is designed by a design team and confirmed with the governance committee of the company. Afterwards, the service or API is implemented by another team. At the end, the architect is not able to match design and implementation together, since project work evolves over time and naturally adapts to changes which are not covered by the delivered documentation.

Another problem is addressed by challenge *C5*. Generally, if an external customer would like to have an API from the industry partner, the customer normally would not know, how he could contact the company. The only way is currently through the customer's own business network.

In challenge *C6*, the industry partner states that there is no API-specific process at the moment. The industry partner uses an existing generic process for the internal API development. That process is nevertheless not optimized for APIs and requires lots of manual paperwork. One interviewed architect even adds that there is generally no process that can be used for APIs, requested by external customers.

Challenge *C7* describes the difficulties with existing tools. The industry partner mentions that there are some tools available to manage the API development process. Although statuses of the projects can be tracked in form of excel files, the current tool support is overall considered as cumbersome and time-consuming.

The last challenge *C8* relates to the development process of services and APIs itself. Normally, a customer is not directly involved into the service or API development. Hence, the customer does not have the chance to provide early feedback and is only able to see the end result. In fact, this might lead to late change requests by the customer which would automatically lead to longer development cycles.

4.1.2. Success Factors as Requirements

Besides the challenges from the industry partner, further requirements from literature for the Collaborative API Lifecycle Management model are collected and consolidated. For that, (critical) success factors from existing API Lifecycles and similar lifecycle approaches are retrieved. The success factors are also discussed with the industry partner to confirm their suitability. Success factors can be described as *„those few key areas that must be payed special and continual attention to guarantee managerial or organizational success“* (Zhang et al. 2013). In total, four areas for lifecycles are considered which have its source mainly in academic literature, but also in white papers, published by leading industry players in the area of API Management and API Lifecycle Management. The leading industry players are identified with the help of the magic quadrant for Full Life Cycle API Management by (Malinverno and O'Neill 2018; Malinverno and O'Neill 2016). The examined areas cover the following:

1. API Management and API Lifecycle Management
2. Product Development Lifecycle
3. Service Management Lifecycle
4. Agile Software Development Lifecycle

The product development lifecycle is chosen, due to the fact that APIs can also be treated as individual products. Besides, an API has strong similarities to a service and is also more or less a piece of software. Because of that, success factors from the service management lifecycle and the software development lifecycle are included into the requirements research. Table 4.2 shows the consolidated results for collecting API Lifecycle Management requirements. Generally, the success factors are chosen based on some criteria. If a success factor is from a literature source about API Lifecycle Management or API Management, then it is automatically included into the requirements list. In case of other examined areas, further criteria are considered:

- Is the success factor one of the top ones in the examined area?
- Is the success factor applicable to other lifecycles?
- Is the success factor realizable?
- Is the success factor mentioned by multiple literature sources?
- Is the success factor applicable to existing API Lifecycles of leading industry players in the area of API Management or API Lifecycle Management?

The collected requirements can be divided into four categories, which are business, organizational, process and technical requirements. In the following, the requirements are briefly described.

Table 4.2.: The requirements for the Collaborative API Lifecycle Management model

	Requirement	**Source**
	Business Requirements	
R01	Strategy and process alignment	(Mulesoft 2014; Zhang et al. 2013)
R02	Provide easy access for users	(Mulesoft 2014; Apigee 2016a)
R03	Achieve stickiness of ecosystems and project champion	(Mulesoft 2014; Vukovic et al. 2016; W.-G. Tan et al. 2009)
R04	Clear business strategy, goals and objectives	(Mulesoft 2014; Cooper and Kleinschmidt 1995; W.-G. Tan et al. 2009)
R05	Adapt to enterprise's level of digital maturity	(Vukovic et al. 2016)
R06	Suitable market environment	(González and Palacios 2002; Cooper and Kleinschmidt 1995)
R07	Delivery strategy	(Apigee 2016a; Chow and Cao 2008)

Table 4.2.: The requirements for the Collaborative API Lifecycle Management model (Cont'd)

	Requirement	Source
R08	Short development time and short time-to-market	(González and Palacios 2002; Cooper and Kleinschmidt 1995)
	Organizational Requirements	
R09	Top management support	(González and Palacios 2002; Zhang et al. 2013; Cooper and Kleinschmidt 1995; W.-G. Tan et al. 2009)
R10	Support for agile team, flexible processes and change management	(Zhang et al. 2013; Chow and Cao 2008; Apigee 2016a)
R11	Clear project roles	(Zhang et al. 2013)
R12	High quality of team capabilities and team synergies	(Cooper and Kleinschmidt 1995; Chow and Cao 2008; W.-G. Tan et al. 2009)
	Process Requirements	
R13	Interdepartmental cooperation and communication	(González and Palacios 2002; Cooper and Kleinschmidt 1995; W.-G. Tan et al. 2009)
R14	Alignment of design and requirements	(Zhang et al. 2013)
R15	Service Level Management	(Zhang et al. 2013)
R16	Project Management	(Zhang et al. 2013; Chow and Cao 2008; W.-G. Tan et al. 2009)
R17	Governance support	(Apigee 2016a; W.-G. Tan et al. 2009)
R18	Divide process into stages	(Zhang et al. 2013)
R19	Customer involvement into the process	(González and Palacios 2002; Chow and Cao 2008)
	Technical Requirements	
R20	Design for user experience (UX) and developer experience (DX)	(Mulesoft 2014; Apigee 2016a)
R21	Provide reusability of product	(Vukovic et al. 2016)
R22	Support continuous integration and continuous improvement	(Vukovic et al. 2016; Zhang et al. 2013; Apigee 2016a)
R23	Ensure security	(Vukovic et al. 2016; Apigee 2016a)
R24	High product and service quality	(Vukovic et al. 2016; González and Palacios 2002)

Table 4.2.: The requirements for the Collaborative API Lifecycle Management model (Cont'd)

	Requirement	Source
R25	Drive end-to-end visibility	(Apigee 2016a; Zhang et al. 2013)
R26	Interactive documentation with self-service capabilities	(Apigee 2016a)

The business requirements focus on strategical factors. In general, a company should always set a clear goal and scope for its business and API strategy (R01, R04). Another important point is that companies need to adapt to different maturity levels (R05). There are companies who are already longer on the API market than others and hence, are more experienced. A company, who just started with API experimentation, need to carefully adapt its strategy to its own maturity level (Vukovic et al. 2016). Related to this, is also the API delivery strategy (R07). Companies need to think about a suitable way to deliver its APIs to the API consumers. For instance, *Apigee 2016a* suggests to use a layered API delivery strategy which *„abstracts the underlying complexities and dependencies of APIs and [. . .] accelerates [the] delivery [. . .]"* (Apigee 2016a). Furthermore, it is important to find the right market segment from which the company can profit the most (R06). This generally includes that the products need to be delivered fast on the market, because *„early product introduction improves profitability [. . .] and allowing development and manufacturing cost advantages"* (González and Palacios 2002) (R08). Another requirement is to build a community, since API Management recommends that APIs should be listed in a catalogue for easy access (R02). This catalogue is often in form of an API portal which has self-service capabilities and aims to keep API consumers interested in the offered products and services. This can be also described as to „stick" API consumers inside the API ecosystem of the company (R03).

The second group of requirements deals with organizational factors or rather with people. For a successful API strategy it is important that managers need to be included into the process (R09). Early and constant input from managers help to succeed the overall API Lifecycle process. Besides, there should be clearly defined roles and the company should invest into their employees' skills (R11, R12). This automatically includes that teams need to be agile and adaptable to the factor „change" (R10).

A further group of requirements concentrates on the process for a lifecycle itself. It is common that lifecycles should be divided into stages to reduce complexity and to be able to efficiently organize the process (R18). The process should pay attention to SLAs, project management and governance support, in order to ensure quality and consistency (R15,

R16, R17). Generally, *„there should be no distinction between [. . .] APIs“* (Apigee 2016a) with regard to their visibility and discoverability (Apigee 2016a). Another requirement for the API Lifecycle is that it needs to support collaboration to achieve alignment between different parts of the lifecycle process (R13, R14). This particularly includes that customers have to be involved into the lifecycle to get early feedback and adapt to occurring changes (R19).

The last set of requirements addresses technical factors. Firstly, APIs should be designed in a way to be reusable and easily consumable by API consumers (R20, R21). This inherently includes that the API design needs to achieve good developer experience (DX). The word „developer experience“ is derived from UX. However, compared to UX, DX not only covers a good user interface experience, but also an appropriate and efficient use of the product or service itself (Fagerholm and Munch 2012). Often related to DX is the developer portal which provides an interactive documentation with self-service capabilities to developers (R26). In general, a portal provides the advantage to serve customers through one single entrance point. A further technical requirement aims at overall high product and service quality (R24). This includes the usage of agile tools like CI/CD for continuous improvement (R22). More importantly, the company needs to ensure security (R23). When developers for example deploy APIs in the public cloud and *„neglect to deploy common API security standards or consistent global policies, they expose the enterprise to potential security breaches“* (Apigee 2016a). The last missing requirement addresses the overall transparency of the lifecycle (R25). It is important to analyze performance bottlenecks as well as to measure usage and adoption of the APIs, in order to continuously improve the company's service.

4.2. API Lifecycle Model

The previously presented requirements build the basis for constructing the conceptual model. The coming section presents the resulted Collaborative API Lifecycle Management model that is formed with the help of academic literature, the foundations of this thesis and further expert interviews with the industry partner. To begin with, an overview about the designed conceptual model is given which is followed by a description of single components like the layers, the roles, the artifacts and the activities.

4.2.1. Overview

Overall, the designed conceptual model for Collaborative API Lifecycle Management has both a service focus as well as a product focus. Figure 4.1 illustrates the Collaborative API Lifecycle Management model, included into the envisioned Full Lifecycle API Management ecosystem. The left side of that presented model depicts an inflow of customers who need an API. The customers can be distinguished between internal employee, partner or external user. Depending on the type of customer, the respective API type (internal, partner, external API) is defined. The need of the customer for an API goes through the CALM model which is located in the center of the ecosystem. The CALM model produces the requested API which then can be used by the customer or rather API consumer to build new innovative businesses. New innovative businesses could be a mobile app, a web app or other areas, as shown in the outflow of the ecosystem.

As shown in the ecosystem, the core of that model is presented in the center which is the API Lifecycle Management. Generally, the API Lifecycle Management is depicted in the innermost layer as a blue box. Belonging to the CALM model are several layers which surround the API Lifecycle Management and provide guidance for the lifecycle. Furthermore, the CALM model consists of a variety of involving people, created artifacts and activities performed by the different roles in the single stages of the API Lifecycle. The following sections break the CALM model into its single components and describe the details.

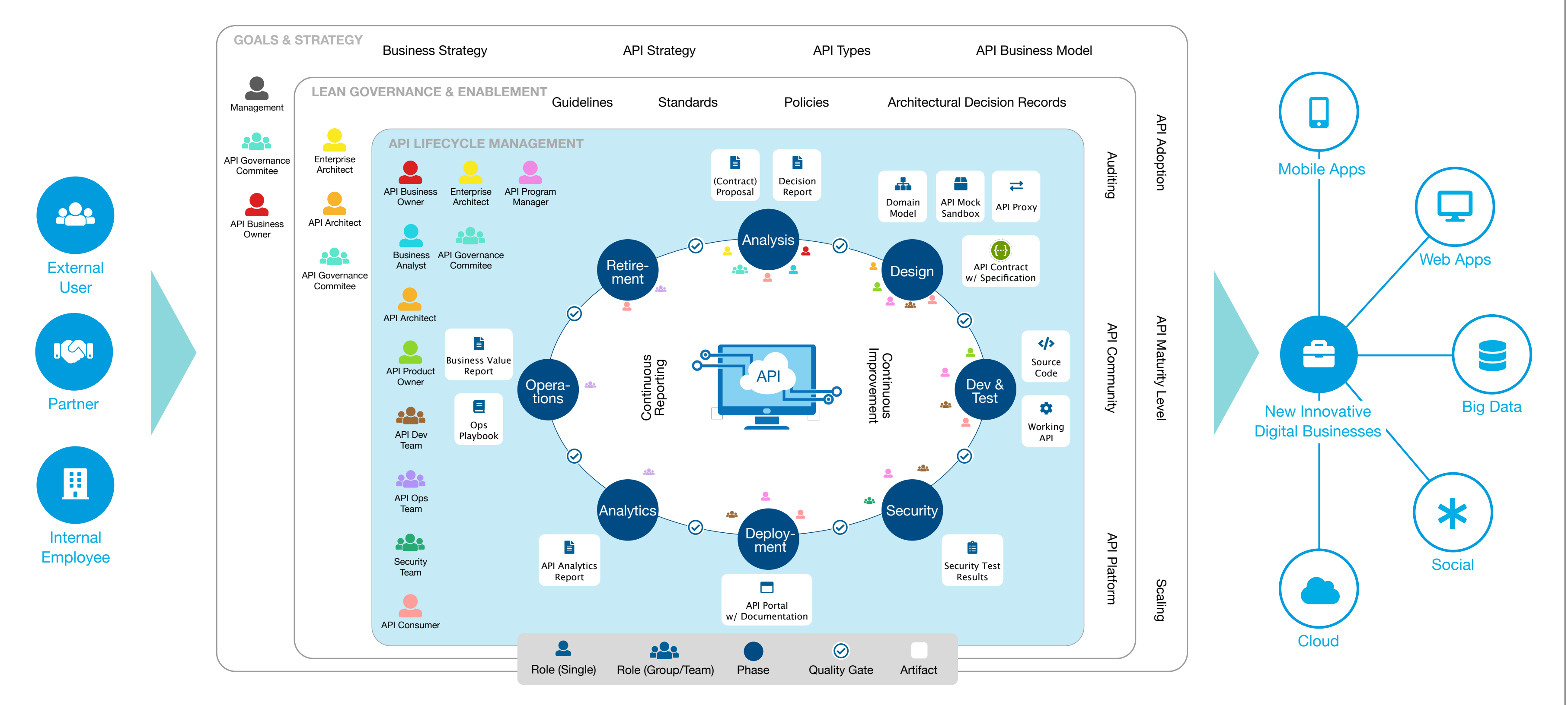


Figure 4.1.: The vision of the overall full lifecycle API Management model and the included model for Collaborative API Lifecycle Management (CALM)

4.2.2. Layers

The CALM model includes in total three layers. The outermost layer defines the goals and strategy of the API provider. This layer ensures the direction and leads the API provider to its goals. As shown in the layer, there are several factors that need to be paid attention to. One factor is that the company needs to define its API strategy. This API strategy needs to align with the business strategy to achieve highest business value. Related to the strategy is the focus on the API types. Generally, it should be a mix out of public and private APIs to gain the most potential on the API market. Another factor to be considered, is the API business model. The API provider needs to adopt the right model for monetizing its APIs and to be successful in the API economy. Common business models that are currently applied on the market are freemium or pay-per-use (Iyengar et al. 2017). Overall, the ultimate goal of an API provider is to achieve high API adoption by API consumers. To achieve that goal, the API provider needs to constantly optimize its API Management. This includes to be aware of the maturity level of the API programs and further, how to scale the API business to provide a variety of APIs and satisfy the different needs of API consumers.

The next inner layer deals with governance and enablement. To offer a variety of APIs, they need to be managed in some central place. A common way is to develop an API platform which is often represented as the API portal. A typical API portal offers API consumers the facility for self-service, including developer onboarding, app registrations and API documentation (De 2017). With the help of the API portal, it is also possible to create a community of API consumers. The better the experience in the portal with APIS, the more likely more API consumers will adopt them. To ensure consistency in terms of API visibility and discoverability, governance need to be introduced. Since too much governance limits API consumers and too less governance hinders API reusability, the CALM model considers only the most needed governance regulations which is also termed as lean governance. Besides basic governance elements like providing guidelines, standards and policies, architectural decision records (ADR) should be also included. ADRs document the decisions for the usage of a certain technology or tool. The company *ThoughtWorks* proposes a model which is known as *Technology Radar* to visualize ADRs and allocate a certain technology to a specific adoption level in the circular radar model (Parsons et al. 2017). All the governance elements should be accessible to API consumers and providers when designing and developing the API. Furthermore, automatic governance test mechanisms need to be introduced for auditing and secure consistency.

The last layer is represented by the API Lifecycle Management and depicted in the center of the CALM model. The API Lifecycle consists of different stages. It starts with the

Analysis stage, goes clockwise until it ends with the *Retirement* stage. Each stage includes a number of interacting people and various artifact. The roles collaborating in a stage can be seen by matching the color of the roles next to the stage to the ones on the left side of the API Lifecycle Management layer. After each phase there are quality gates which are function as checkpoints. The quality gates ensure that before a stage switches to the next one, it fulfills a list of quality standards. By that, the quality of the API programs is ensured over the whole lifecycle. Another point to be mentioned is that once an API reaches a certain stage, it does not mean that activities from previous stages cannot be performed again. In fact, the whole lifecycle is considered as flexible and agile to achieve continuous improvement. The stage of an API only reflects the latest one, in which the API performs activities initially. Further details to the roles, artifacts and activities are examined in the coming sections.

4.2.3. Roles

In each layer of the CALM model, there are several roles involved. Each role collaborates with other roles in that layer to realize a part of the whole API Lifecycle. A role can interact in multiple layers and can make decisions about layer-specific topics that was presented in the previous section. Most of the roles are located in the innermost layer, the API Lifecycle Management. Besides the different roles belonging to the API provider, the API consumer is also involved along the whole end-to-end lifecycle for an API. Table 4.3 gives an overview about the responsibilities and functions of each role. Most of the shown information is retrieved from (De 2017; Ravichandran et al. 2016) and additionally confirmed with the industry partner. It should be noted that further roles can be included into the lifecycle. However, the current version of the CALM model covers only the necessary ones and those who are expected by the requirements.

Table 4.3.: The roles of the CALM model

Role	**Description**
Management	Leads the company towards the API strategy
API Governance Committee	„Ensures that the process is followed, criteria are met, and quality is maintained“ (De 2017)
Enterprise Architect	Responsible for governance related topics, screening and reviewing API proposals
API Business Owner	„Responsible for establishing and validating the business needs of the API and the requirements for approval of funding“ (De 2017)

Table 4.3.: The roles of the CALM model (Cont'd)

Role	Description
API Program Manager	„Responsible for the overall program delivery of the APIs“ (De 2017)
API Architect	„Responsible for the technical architecture of the API solution“ (De 2017)
Business Analyst	„Gathers the business requirements for API enablement and identifying the services to be exposed as APIs“ (De 2017)
API Product Owner	„Responsible for interfacing with various API delivery teams to ensure the quality and delivery of the APIs“ (De 2017)
API Dev Team	Responsible for the overall API delivery, including design, development and testing of the API
API Ops Team	Responsible for the deployment of API, monitoring analytical metrics (adoption, usage, performance, etc.), developer onboarding and resolving issues of the API
Security Team	Responsible for the security of API and performs penetration tests before deploying APIs into production
API Consumer	Uses the API for own apps or other new innovative businesses

4.2.4. Artifacts

Along the whole lifecycle, a number of artifacts are created within or at the end of a stage. The artifacts can have different purposes. There are artifacts which record and document decisions, provides updates about the stage's status or are just used as inputs for further artifacts, resulted in subsequent stages. Table 4.4 summarizes each artifact's importance in the API Lifecycle.

Table 4.4.: The artifacts of the CALM model

Stage	Artifact	Description
Analysis	(Contract) Proposal	Contains the details of an API like name, description, pricing, SLAs, files, etc.
	Decision Report	Records and documents the decision for choosing a proposal

Table 4.4.: The artifacts of the CALM model (Cont'd)

Stage	Artifact	Description
Design	Domain Model	Builds the basis for creating APIs e.g. in form of micro services
	API Mock Sandbox	Adds dummy values to mock an API based on a provided specification
	API Proxy	Used in connection with the API mock and functions as communication middle layer between front-end and back-end
	API Contract with API Specification	Represents the actual contract containing the requirements, design, API specification, etc. before implementing the logic
Dev&Test	Source Code	Represents the logic of an API
	Working API	Represents the API end product
Security	Security Test Results	Informs about vulnerabilities before API can be deployed in production
Deployment	API Portal with Documentation	Puts the API and its documentation into the API Portal
Analytics	API Analytics Report	Reports adoption metrics, performance metrics or other analytical KPIs for improvement and monitoring the status
Operations	Business Value Report	Informs about the overall monetization results of all API programs
	Ops Playbook	Represents the „technical instructions“ to maintain the deployed API
Retirement	-	-

4.2.5. Activities

Various activities are performed in each stage by the different roles of the API provider and the API consumer. Table 4.5 gives an overview about possible activities inside a certain stage. The activities result from expert interviews, own industry experience and literature sources like De 2017. The shown activities are not complete and show a first set of possible actions. The shown activities reflect the API provider perspective.

Table 4.5.: The activities in each stage of the CALM model

Stage	Activity
Analysis	Submit API proposal by API consumer (new API request, change request) Define of quality and governance requirements for API e.g. SLA, NFRs Analyze API proposal (business alignment, feasibility) Get two step approval for API (governance and funding) Define team roles for API programs
Design	Get governance requirements for API (standards, guidelines, policies, coding conventions, etc.) Conceptualize granular design of API with API consumer (data model, API specification, etc.) Generate DevOps environment automatically (repository, unit tests, documentation, API gateway, API proxy, API mock, etc.)
Dev&Test	Implement business logic of API Test business logic of API with real data (unit test, integration test, etc.)
Security	Perform critical security test (penetration test, load test, performance test, authentication and authorization test, etc.)
Deployment	Deploy production-ready API Publish documentation to API portal Get approval for legal requirements Get approval for marketing requirements (monetization, corporate design, etc.)
Analytics	Define API metrics to be monitored e.g. health status, traffic consumption, traffic spikes, performance, API adoption, etc.
Operations	Provide maintenance and solve issues (bug fixing, logic improvements, etc.) Manage API monetization, developer onboarding, API scaling
Retirement	Manage change notifications (deprecation, retirement) Disable subscription of deprecated and retired APIs

4.3. Collaboration Features

Since collaboration plays an important role in the CALM model, the following section describes the chosen features for the design artifact. Collaboration generally enables to align and synchronize all participants in the API Lifecycle. It helps API consumer and provider to efficiently and effectively work together on one common medium and to reach a common goal. This common medium is an API proposal. The idea is that all new requested APIs are organized in proposals. All related information to an API is grouped at one place.

As described in section 2.4.2 and shown in Table 2.2, there are four dimensions for collaboration tools. For the design artifact, the 4th dimension contains the most suitable tools for the prototype. To achieve flexibility, it is important to choose tools that neither limit the place nor the time. For the prototype, a voting system and commenting system is chosen. In addition, API mocking is added. Figure 5.3 shows a mock-up for the *API Proposal Details View* with the collaboration features.

The voting system creates awareness. It helps to include the outside-in-thinking concept, since all proposals are currently visible to all users of the system. The more people vote for an API proposal, the more the importance and need on the current market. This makes the screening of proposals easier for the API provider.

The commenting system follows the concept of a web blog. The API proposal which equals to the web blog is extended by a commenting system. This is especially useful for exchanging communication and cooperation between API consumer and provider.

The last collaboration feature is API mocking. While the two previous features are directly or indirectly listed in the collaboration tool classification as depicted in Table 2.2, the API mocking feature is not. However, API mocking enforces collaboration in the API context. Since API First is used for the design artifact, API mocking can be included into the prototype. Based on the specified details of an provided API specification, a mock for the described API can be generated which enables API consumer and provider to work independently. API provider can implement the logic of the API, while the API consumer can use the API with dummy values to integrate it into the his own business like a mobile app, web app or others. Whenever the API specification is updated, both parties can be notified. Combined with further technical tools, this feature helps to synchronize API consumer and provider easier.

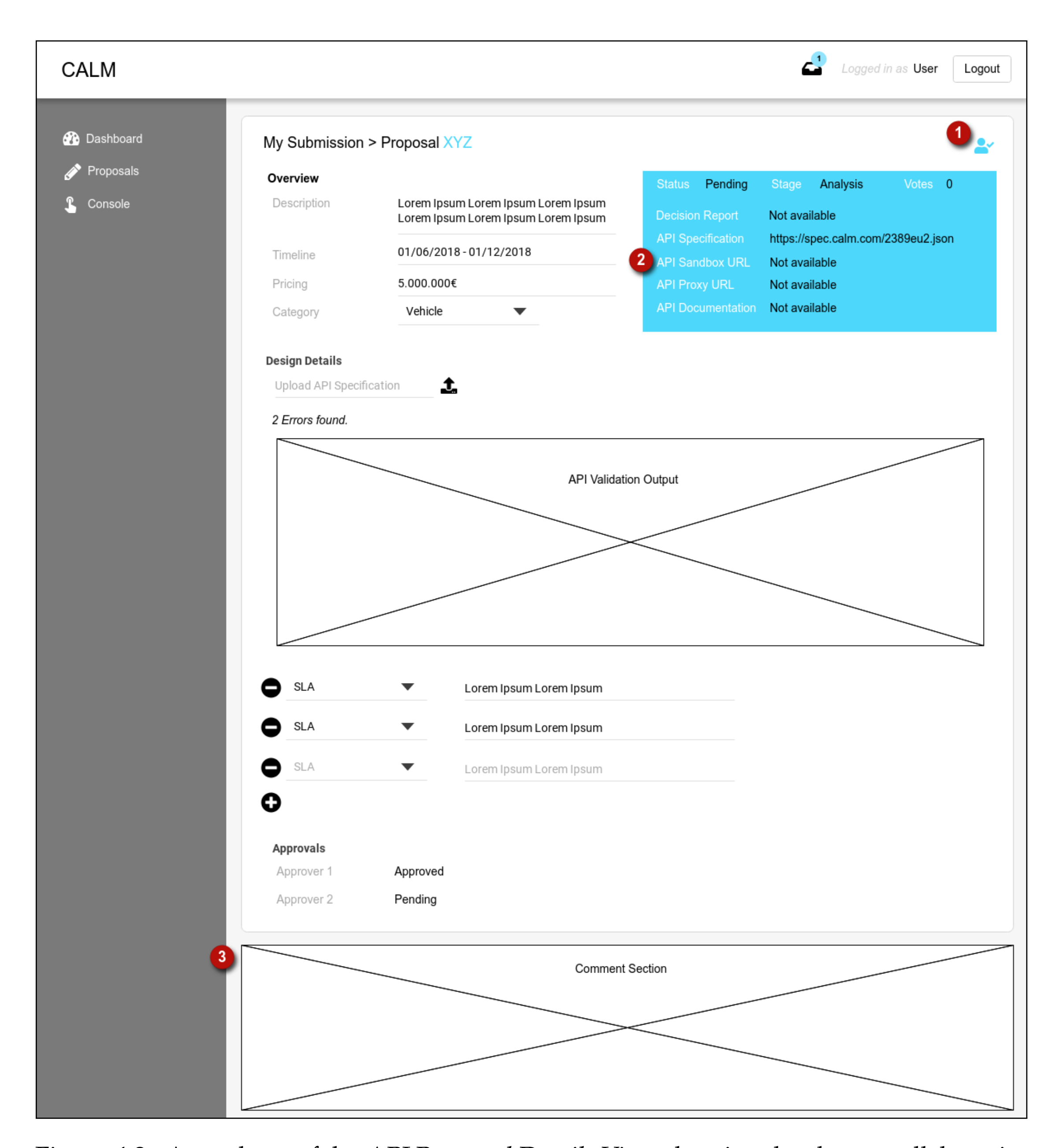


Figure 4.2.: A mock-up of the *API Proposal Details* View showing the chosen collaboration features: (1) Voting System, (2) API Mock, (3) Commenting System

4.4. System Design

To technically support the CALM model, a design artifact in form of an instantiation is needed. In agreement with the industry partner, the current prototype version covers only the first two stages from the conceptual model as proof of concept. Before the IT artifact can be implemented, the software design needs to be prepared first. Therefore, the following section starts with an overview about use cases for the overall API portal and for the current version of web application prototype. Subsequently, the activity flow for the first two stages is presented as a general process overview. Furthermore, the role-based access control (RBAC) model and the data model for the prototypical solution are introduced.

4.4.1. Use Case Diagram

Several use cases for the envisioned API portal are identified. The use cases are mainly derived from the CALM model. Figure 4.3 depicts a use case diagram for the overall vision for the API portal.

The use case diagram covers a number of various actors who are potential users of the API portal. The actors reflect the same roles as the ones from the CALM model. They can be divided into two groups which are API consumers and API providers. While the left side of the use case diagram shows API consumers, the right side of the use case diagram shows API providers. An API consumer can be an external user, a partner or an internal employee. The API provider side only includes roles like API program manager, enterprise architect, API architect, API product owner, API development team, API operations team and security team. Not all roles from the lifecycle model for APIs are active users of the envisioned portal.

The center of the use case diagram shows the CALM prototype as subsystem with several use cases depicted as bubbles. In general, the use cases can be grouped into eight key functionalities. The key functionalities are depicted on the top left of the use case diagram and include project management, proposal management, API subscription management, governance management, DevOps management, security, analytics and version management.

The first functionality covers *project management* features. The general idea of that functionality is that with the help of the API portal, it should be possible to get an overview of all APIs in development and to trace each API program's progress individually. Also, a feature for automated report generation should be included into the API portal to record

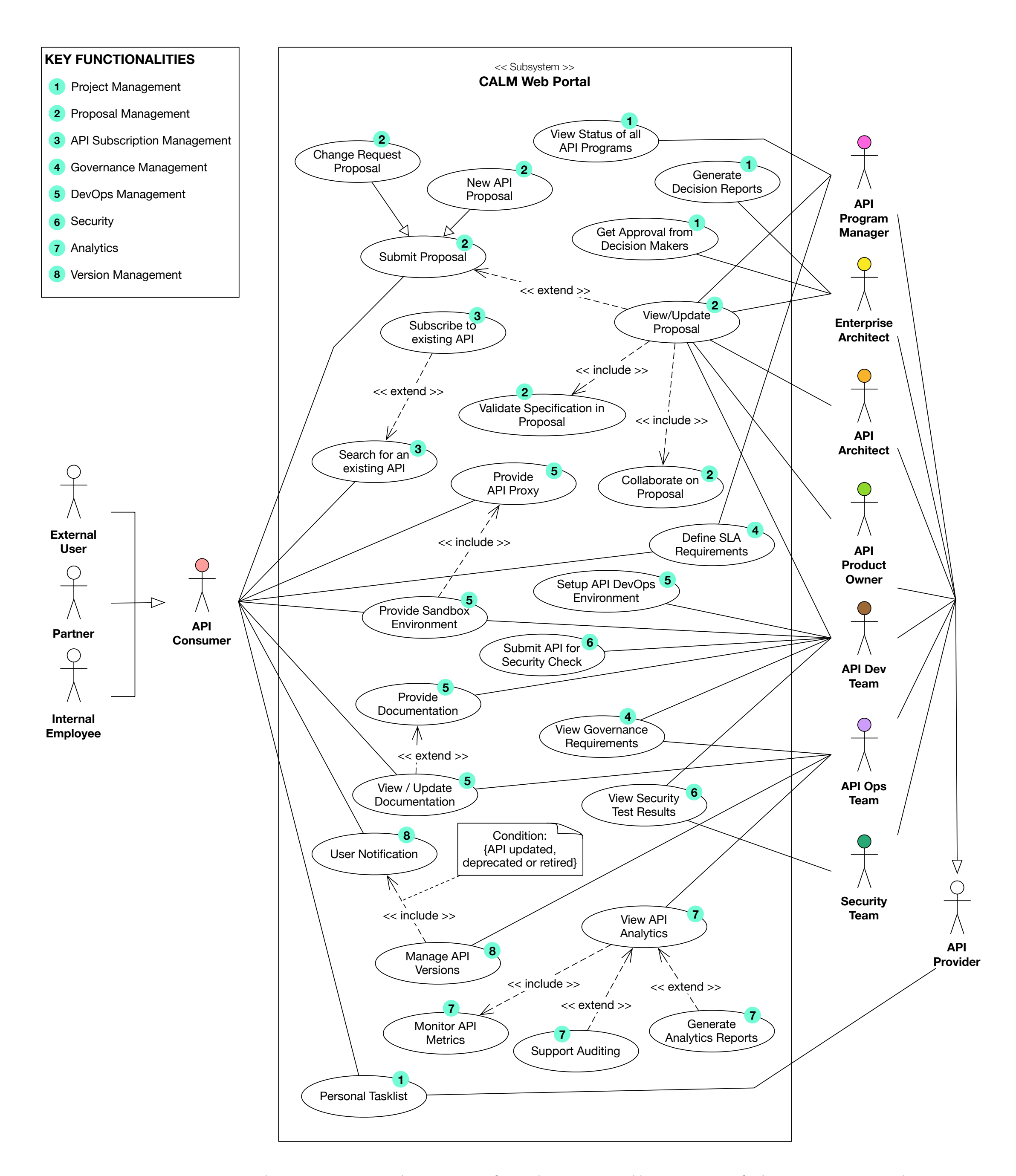


Figure 4.3.: The use case diagram for the overall vision of the API portal

decisions. Furthermore, during the lifecycle of an API, several approvals from decision makers need to be requested. This could include decisions for instance about funding, governance conformity, marketing and legal factors. A personal task list is another minor feature which intents to assign tasks to certain people or group of people.

The collaboration of API consumer and provider is based on one common medium. This medium is an API proposal. Because of that, several features for the *proposal management* are planned. The *proposal management* includes submitting, viewing and updating API proposals. Also, further collaboration features beside the ones presented in 4.3 help to form the proposal details. In addition, API proposals can be attached with files like an API specification. To validate the API specification, a linter feature is included to syntactically check for violations against guidelines, proposed by the API provider.

Once some APIs are deployed and ready to be used by API consumers, the API Portal needs to provide users the possibility to subscribe to APIs. The subscription would generate tokens for the consumers to access and consume the APIs. Furthermore, users should be able to search and filter for existing APIs by certain criteria. Those features are grouped into the *API subscription management* functionality.

The *governance management* functionality groups features to guarantee the consistency of APIs. The API portal needs to set governance mechanisms along the API Lifecycle process to ensure quality of the products and services. For that, API consumers should be able to define SLAs. Furthermore, both API consumer and provider can view the company's governance guidelines inside the API portal which for example helps them to define the API specification.

Besides, the *DevOps management* functionality targets the support of the API portal with development tools. In fact, the API portal aims at an automated approach for setting up the development environment which covers elements like an API documentation, API mock, CI/CD, repository or further features.

The following two functionalities deals with *security* and *analytics* related features. Before an API can be deployed live into production, it needs to go through several security checks. A possible use case would be to include the security test results into the API portal to make them visible to API consumers and providers. After an API went live into production, the service should be continuously improved. For that, several analytics features support the monitoring of the API performances and also their adoption by the API consumers.

The last functionality is the *version management*. The main goal of that functionality is to manage the different versions of an API. Furthermore, whenever a new version is about to be deployed, the API consumers should get a notification. In general, all consumers

should be informed about any changes not only in production-ready APIs, but also in API proposals.

The design artifact in form of a web application prototype does not include all presented use cases from Figure 4.3. Only a set of use cases is extracted for the prototypical implementation. The use cases that need to be realized are depicted in Figure 4.4.

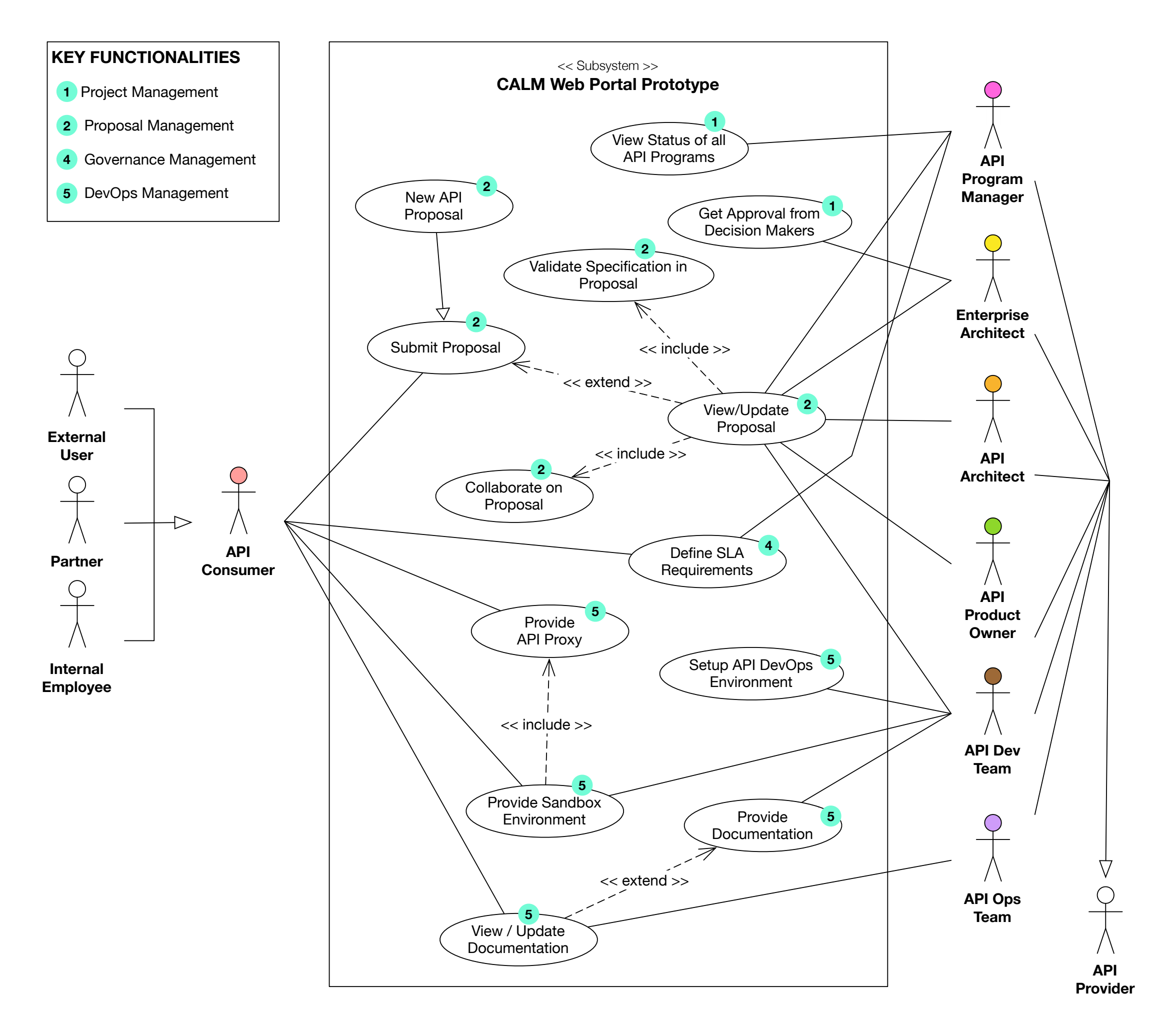


Figure 4.4.: The use case diagram for the web application prototype

4.4.2. State Machine Diagram

To understand the overall activity flow in the first two lifecycle stages *Analysis* and *Design*, the following section explains the chain of actions in form of a state machine diagram. The state machine diagram is illustrated in Figure 4.5. The lifecycle stages are displayed in parallel to show the transition point between the *Analysis* and the *Design* stage, depicted by a vertical dashed line.

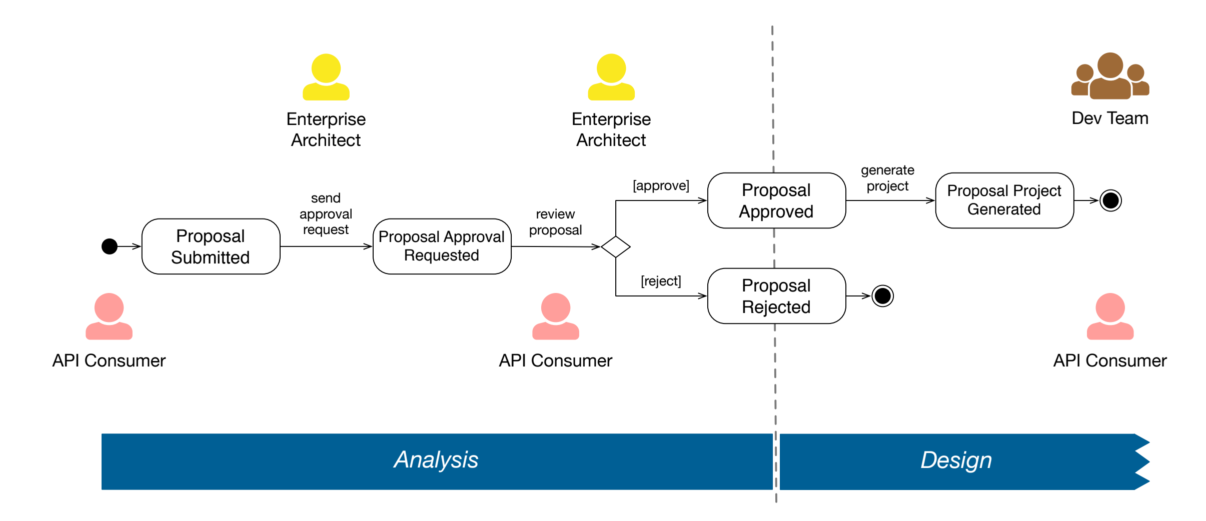


Figure 4.5.: The activity flow of the first two stages of the web application prototype in form of a state machine diagram

In general, the prototypical solution supports the CALM model with collaboration features and an acceleration approach. As described in section 4.3, the prototype uses a commenting system and a voting system. The commenting system can be used throughout the whole lifecycle, whereas the voting system is mainly used at the beginning. Furthermore, an API mock is used not only to strengthen collaboration, but also to accelerate the process. The API mock separates dependencies between API provider and API consumer. To enable the API mock, an API First approach is needed. This includes the preparation of an API specification.

As already mentioned, API proposals build the common medium on which API provider and API consumer can collaborate with each other. While the API provider is shown on top of the state machine diagram, the API consumer is shown below. The process starts with the API consumer who submits an API proposal with its details like API name, description and pricing. In addition, he can optionally attach files to the proposal like an API specification or other related files.

After receiving the API proposal, the API provider side, represented by an enterprise architect, screens through the API proposal and checks, if it aligns with the API provider's

interests. To include the outside-in-thinking, all submitted proposals are publicly visible to all users of the web application prototype. With the help of the voting system, all users can indicate their interest in a certain API. If a proposal holds a big amount of supporters, then it will catch the attention by the API provider easier. In case of a positive interest from the API provider side, the EA would send an approval request for a proposal.

This particular step sends two approval requests to two different decision makers. On the one hand, the API governance committee approves for the general alignment with the API provider's strategy. On the other hand, an API business owner approves for a positive funding of the API program development. Simultaneously, API consumer and provider iterate over the API proposal's details and form a valid API specification that is included into the API proposal. To ensure that an API proposal is valid, linting tools are used to syntactically check for governance violations.

Once an EA receives two positive approvals and he assess that the current details of the API proposal are sufficient, he would mark the proposal as approved. Subsequently, the API proposal is automatically prepared for the next stage which include the automated generation of the development environment (repository, CI/CD, API documentation, API mock, etc.). At the same time, the EA checks the results of the generated project and hands it over to the next team members. Accordingly, if the EA receives at least one rejection decision, he marks the proposal as rejected and the proposal will be archived.

4.4.3. RBAC Model

Since the CALM model includes various roles, it is necessary to define access control rights and permissions for the people with different activities to ensure security. For that, the *RBAC'96* model, described by (Sandhu 1998), is applied. In general, the *RBAC'96* model differentiates between *users*, *roles* and *permissions*. With regard to the prototypical solution, a *user* equals to a human being, a *role* to a job function and a *permission* to a *„particular mode of access to one or more objects in the system"* (Sandhu 1998). The brief idea of that model is to map *users* to *roles* and *roles* to *permissions*. For instance, a *user* can belong to certain *roles* which have certain *permissions*. In the case of the prototype, a *user* belongs exactly to only one *role*. Furthermore, within *users* can be a role hierarchy. Figure 4.6 shows the role hierarchy model for the prototypical solution.

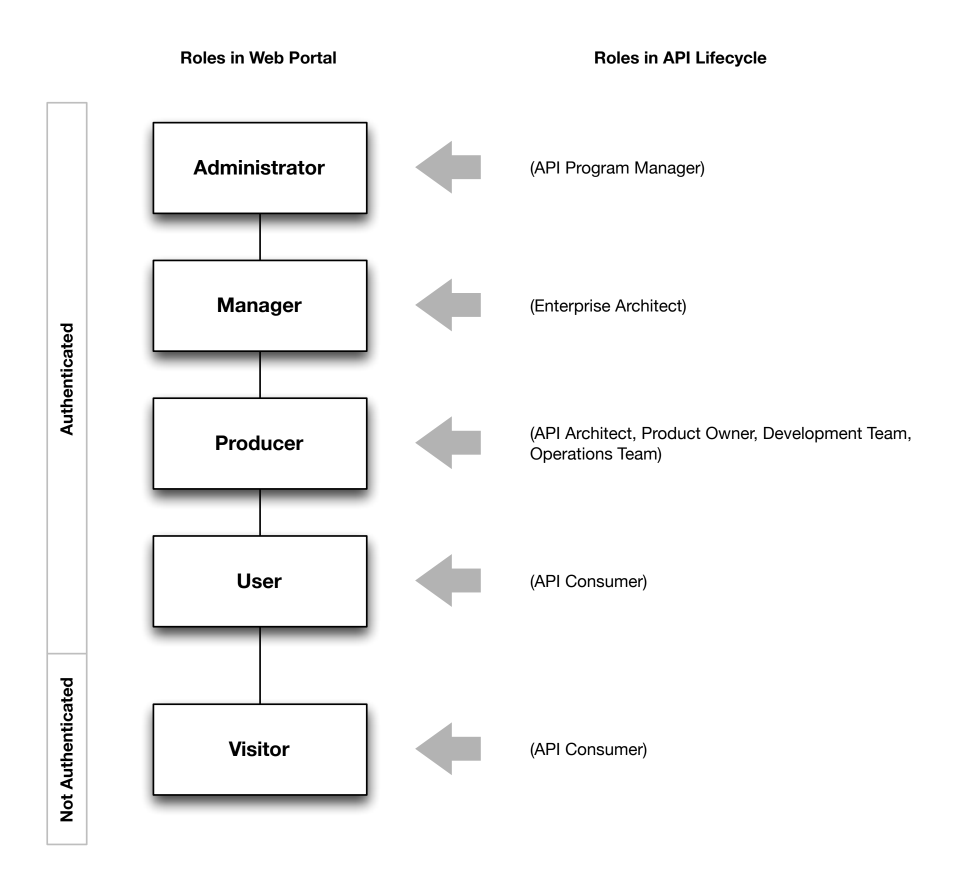


Figure 4.6.: The role hierarchy with role mapping between the CALM model and the web application prototype

The role hierarchy model contains five user roles. The roles from the CALM model can be mapped to a respective one in the prototype. The user role with the most „impacted" permissions is depicted at the top. The impact of the permission rights for the roles lowers gradually from top to down. Besides, the user roles can be separated into authenticated and non-authenticated ones. Non-authenticated roles equal to having no registered

account in the web application prototype. The only non-authenticated role is the *Visitor*. The remaining ones, including *Administrator*, *Manager*, *Producer* and *User*, belong to the authenticated category.

Besides the presented role hierarchy model, a matrix showing the mapping of activity permissions to the user roles is illustrated in Table 4.6. The current prototype version considers in total 14 permissions. Most of the permissions are self-explaining, except for P1 and P2. In case of P1 and P2, the overall idea is that users with a *Visitor* role can only see non-sensitive information of the API proposals like API name and description, whereas the remaining roles can mostly see all information, including pricing and the company proposing the API. In addition, a user with a *User* role can only see all information about an API proposal, if he submitted the proposal by himself which equally means to be the owner of the proposal.

Table 4.6.: The RBAC model for the web application prototype. (R: Role, P: Permission)

	P1: View API Proposal (Parts)	P2: View API Proposal (Full)	P3: Create API Proposal	P4: Update API Proposal	P5: Delete API Proposal	P6:Comment API Proposal	P7: Vote API Proposal	P8: Share API Proposal	P9: Get Approval for API Proposal	P10: Review Proposal	P11: View SLA	P12: Manipulate (Create, Update, Delete) SLA	P13: View Status of all APIs	P14: View Sandbox/API Proxy
R1: Visitor	x					x		x					x	x
R2: User	x	x*	x	x*		x	x	x			x*	x*	x	x
R3: Producer		x	x	x‡		x	x	x			x	x‡	x	x
R4: Manager		x	x	x‡	x†	x	x	x	x†	x†	x	x‡	x	x
R5: Administrator		x	x	x	x	x	x	x	x	x	x	x	x	x

*Role must be owner of the proposal

†Role must have the same domain as the proposal

‡Role must be owner of the proposal or must have the same domain as the proposal

4.4.4. Data Model Diagram

In the following, a brief introduction into the data model is given. As shown in Figure 4.7, the data model can be grouped into four areas, including *web application*, *source code management* (SCM), *CI/CD* and *file server*.

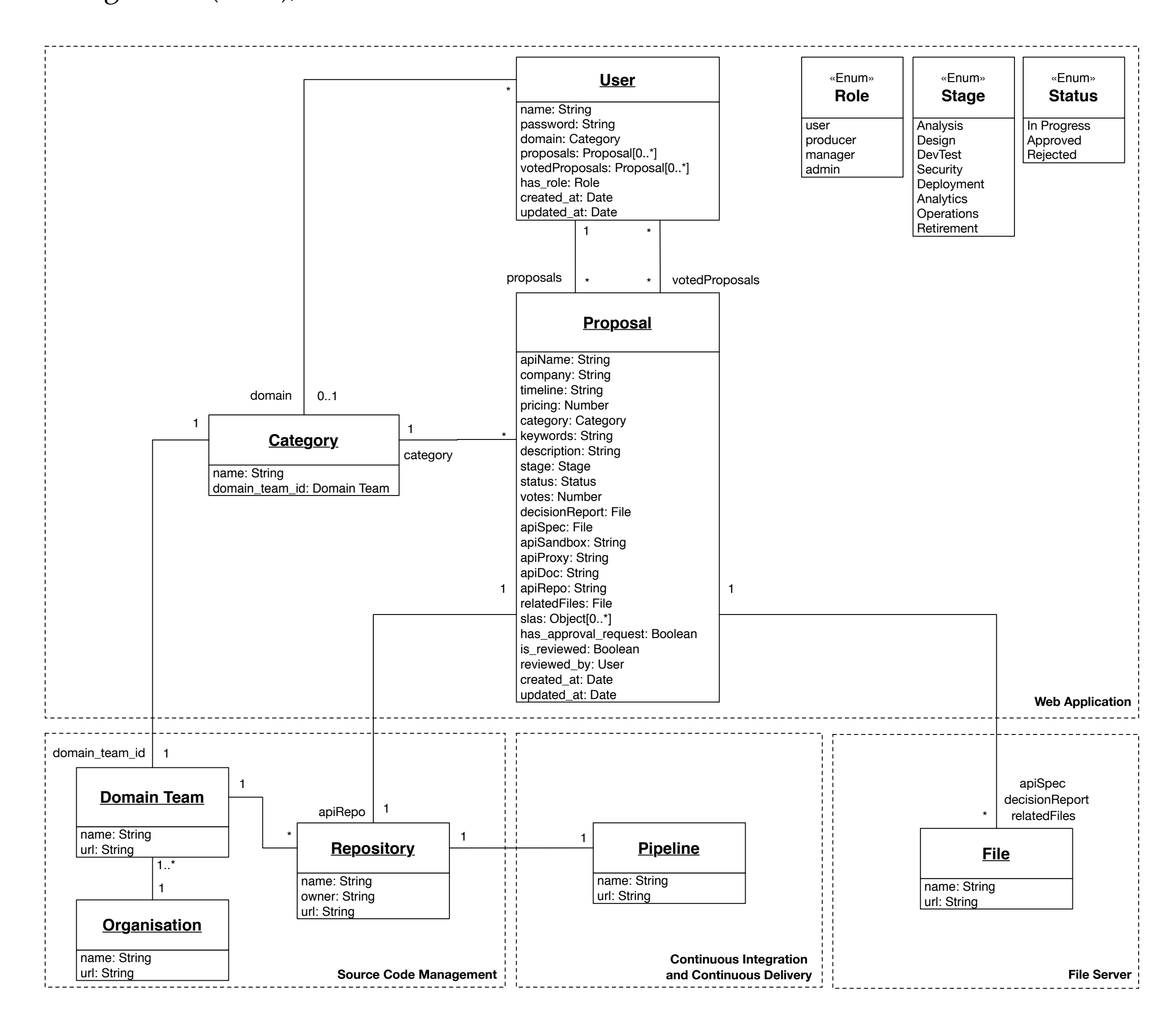


Figure 4.7.: The data model of the web application prototype

All prototype related data collections are grouped in the *web application* area. The remaining areas have a rather supportive function. Particularly, the prototype related data collections consist of a user, a proposal and a category data collection. Each of the collections in the data model holds various attributes. In addition, there are relationships between the data collections. A user holds a list of proposals that are owned by him and a list of proposals that he voted for. Both a proposal and a user belong to a category or rather a domain. An API proposal is connected to a file server to store files like the API specification. Furthermore, a proposal has a repository which is connected to a CI/CD pipeline to enable continuous improvement. A repository is allocated to a domain team in

the *source code management* area. The number of domain teams depends on the number of categories in the web application. All domain teams are part of one organization which is the API provider's company.

5. Prototypical Implementation

After forming the conceptual base for the design artifact, the next chapter gives an introduction into the technological realization of the web application prototype. As previously mentioned, the prototype focuses on the first two stages of the Collaborative API Lifecycle Management. For that, an overview about the general architecture is given which is followed by a short outline of the technological foundations. Afterwards, the front-end and the back-end of the prototypical solution are each elaborated in detail.

5.1. Architecture

The whole architecture of the design artifact follows the design best practices of a microservice architecture. Figure 5.1 shows an overview about the underlying architecture of the prototypical solution in form of a component diagram. The idea of a microservice architecture is to break a monolithic system into small, autonomous and easily manageable services that interact with each other. Furthermore, microservices follow the two principles of loose coupling and high cohesion which are common design goals for all kinds of software development products. Loose coupling means that *„a change to one service should not require a change to another."* (Newman 2015). High cohesion can be described as related behavior needs to be grouped together, while unrelated behavior needs to be separated.

The architecture of the prototype consists of five subsystems, one database and one file server. The first subsystem is the *Web Application Prototype* which at the same time represents the core of the architecture. The *Web Application Prototype* subsystem is divided into a *User interface* (UI) and a *Backend for Frontend* (BFF). The UI is specifically a web UI, serving as the only component of the front-end. Further details are described in section 5.3. The BFF is part of the back-end and serves as a communication middle layer between the UI and various back-end microservices. The idea of the BFF pattern is that for every type of the UI e.g. web, mobile, tablet, etc., a corresponding server-side component, called BFF, is customized. Depending on the type of UI, the BFF then handles different

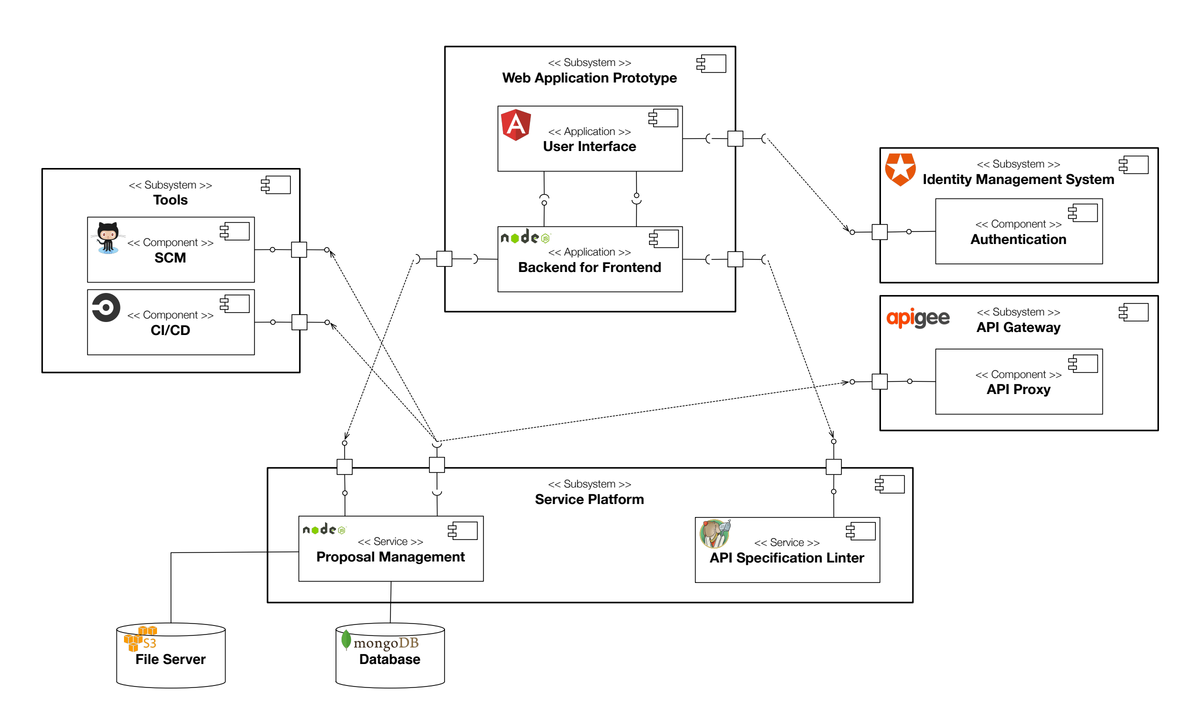


Figure 5.1.: The overall architecture of the prototypical solution for Collaborative API Lifecycle Management

amount of communication intensity with each single back-end service from the *Service Platform* subsystem. Section 5.4 provides further insights into the BFF.

Connected to the BFF, is the second subsystem which is the *Service Platform*. The *Service Platform* subsystem collects all back-end services. Each service provides a certain service functionality. Currently, the prototypical solution makes use of two services, which are the *Proposal Management* service and the *API Specification Linter* service. The *Proposal Management* service handles all information directly related to API proposals. As elaborated in section 4.4.2, API proposals build the common medium for collaboration between API consumer and API provider for the first few stages of the CALM model. Linked to the *Proposal Management* service is also a database for persisting proposal information (see *Data Model* in section 4.4.4). In addition, the *Proposal Management* component handles also the communication to a file server, which stores e.g. API specifications and other files of the API proposals. More details about the *Proposal Management* service are outlined in section 5.4. The overall task of the *API Specification Linter* service is to structurally check API Specification files for violations against predefined governance guidelines. Deeper insights about that service are given in section 5.4.

The remaining three subsystems are the *Identity Management System*, the *API Gateway* and the *Tools*. The *Identity Management System* handles the server-side authentication and provides access tokens to the UI. In other words, it realizes the user login and user registration functionality of the UI. The main function of the *Identity Management System*

is to manage all registered users for *Web Application Prototype* and to set user roles, user permissions, etc. Both the *API Gateway* and the *Tools* subsystems are used by the *Proposal Management* service to enable the automated project generation approach for a successfully approved API proposal. Specifically, the *API Gateway* provides an API proxy which is needed for enabling the API mock (cf. section 4.3). For the current version of the prototype, this API mock feature could not be realized yet and is subject to future work. The *Tools* subsystem consists of a source code management (SCM) system as well as a continuous integration and continuous delivery (CI/CD) system. Both are related to the development environment of an API.

As a final remark, the prototypical solution uses REST as communication basis between the presented components. Furthermore, the provided architecture is considered as a generic one. This particularly means that the used technologies and programming languages for the components of the prototype's architecture are exchangeable with others on the market.

5.2. Technological Foundations

The following section provides a short overview about the used technologies and programming languages for the prototypical realization. The technological details for each single component of the architecture are presented in Table 5.1. As previously mentioned, the chosen technological support for each component is not limited to the displayed ones.

Table 5.1.: An overview about the technological support for each component of the prototype's architecture

	Architectural Component	**Technological Details**	**Reference**
01	User Interface	Angular 5	angular.io
02	Backend for Frontend	NodeJS	nodejs.org
03	Proposal Management	NodeJS	nodejs.org
04	API Specification Linter	Zally	github.com/zalando/zally
05	Identity Management System	Auth0	auth0.com
06	API Gateway	Apigee	apigee.com
07	SCM	Github	github.com
08	CI/CD	CircleCI	circleci.com
09	Database	MongoDB, Mlab	mongodb.com, mlab.com
10	File Server	AWS S3	aws.amazon.com/s3

The prototype generally uses the MEAN stack as coding foundation. The web application makes use of *Angular 5* for the front-end development and *NodeJS* for the back-end development, including the BFF and the *Proposal Management* service. Since the commu-

nication basis is REST, the realization of the API endpoints for the back-end components is done by using *ExpressJS*. *MongoDB* is used for the database and is hosted on *Mlab* for development convenience purposes. The remaining architectural components 4 to 10 are supported by technologies, provided by various companies.

At the very beginning of the prototype's implementation phase in this thesis, *Camunda*[1] was tried to be used for the realization of the *Proposal Management* service. This open-source software tool is a workflow and decision automation platform, supporting BPMN, CMMN and DMMN models, and is rest upon *Java SpringBoot*[2]. The advantage of *Camunda* is that business processes can be automated, analyzed and optimized with visual tools. Since a flexible, agile process for the prototypical solution was needed, BPMN was firstly used as basis, but then replaced by the CMNN modeling language. First initial versions of both business process modeling approaches are attached in the appendix A. The whole solution approach with *Camunda* was later suspended, due to complexity reasons. Nevertheless, this technology is still worth considering for other functional services.

5.3. Web Frontend

After introducing the architecture and the technological foundations for the prototype, the following sections deals with the front-end development details. At first, an overview for the whole realization of the front-end is given. Subsequently, the major realized views of the web application is presented.

5.3.1. Overview

As previously mentioned, the whole front-end architecture is built on *Angular 5*. In addition, the visual representation of the front-end elements is supported by two frameworks, namely *Angular Material Design*[3] (latest version 5) and *Twitter Bootstrap*[4] (latest version 4). The predefined visual elements from the *Angular Material Design* framework build the foundation for the front-end, while only the grid system of the *Twitter Bootstrap* framework is used for arranging the elements in the web browser. An overview about the overall front-end structure is depicted in Figure 5.2.

[1]https://camunda.com/
[2]https://spring.io/projects/spring-boot
[3]https://v5.material.angular.io/
[4]https://getbootstrap.com/

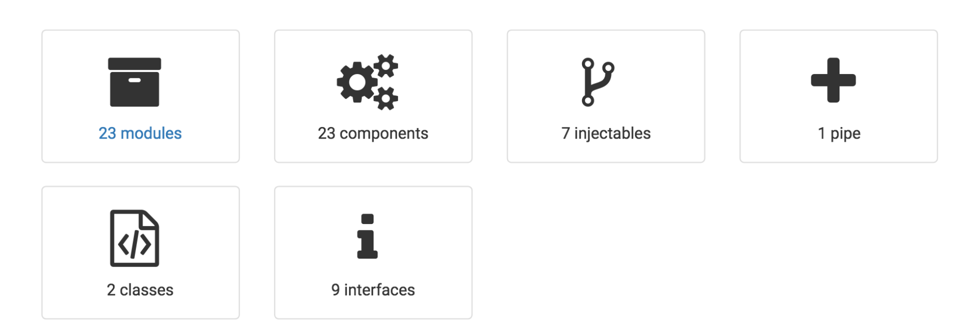


Figure 5.2.: An extract from the prototype's front-end documentation

In general, the realized front-end covers 23 components and each of them is organized in one module. The reason why the components are organized in modules is because of reusability and ease of maintenance. One component can consist of several other components that are harmonizing with each other. All major views of the prototype are for instance based on a big component which includes other smaller components. Besides of components, seven injectables are used for data access, authentication handling and other interaction purposes. The prototype makes also use of a pipe which transforms data into a certain data format for visualizing data through a component. Lastly, two classes and nine interfaces are used for type casting the requests and responses of API endpoints.

5.3.2. Major Views

The four major views of the prototypical web application are elaborated in the following section. The four major views are the *Home View*, the *Proposal Creation View*, the *Proposal Overview View* and the *Proposal Details View*.

Home View

The *Home View* is the first view that a user sees after he successfully logged in into the web application. Figure 5.3 shows an example screenshot of the *Home View*. The general idea of the *Home View* is to have an overview about existing API proposals in the system. The existing API proposals are depicted as white rectangle-shaped boxes. Furthermore, the user can filter for certain proposals, based on predefined categories. The filter is centered in the screen and located above the white API proposal boxes. Another feature of the home screen is that a user can also look into the details of a proposal by clicking on the respective box. In addition, the user has the possibility to navigate to the *Proposal Creation View*, depicted by the blue rectangle-shaped box.

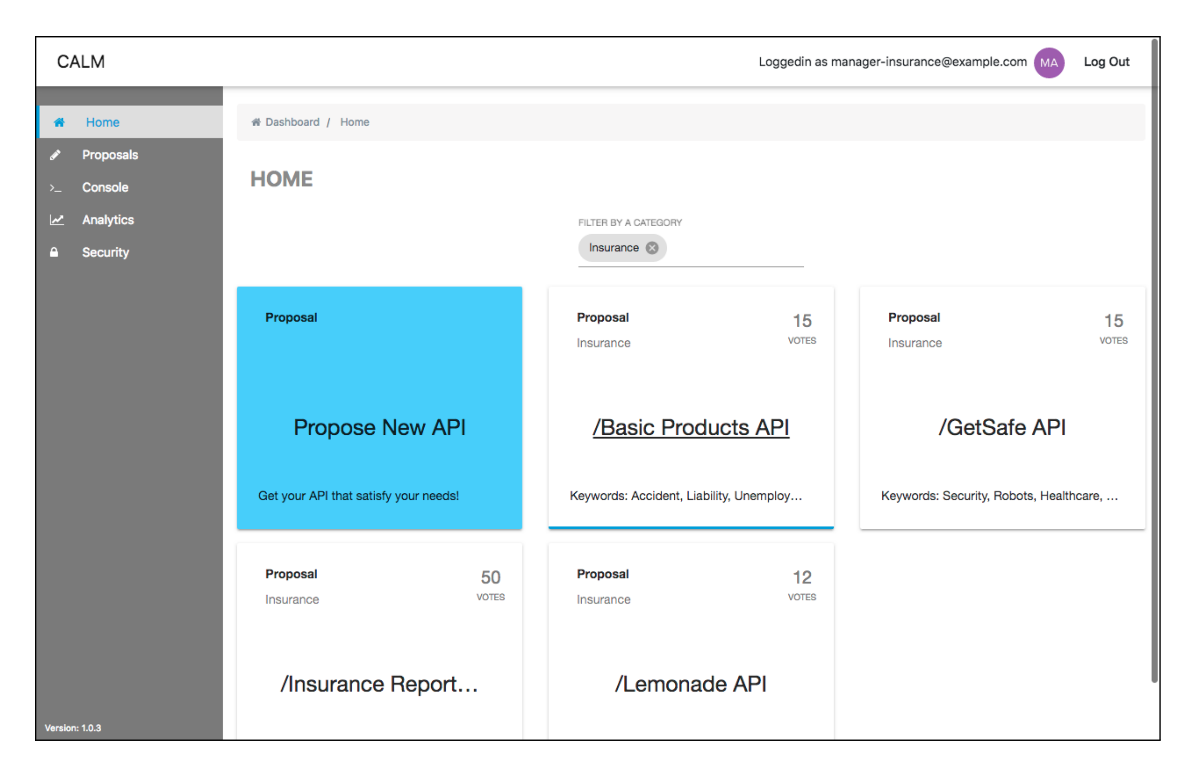


Figure 5.3.: An example of the prototype's *Home View*

Proposal Creation View

In the *Proposal Creation View*, the user is guided through a predefined form to submit an API proposal. The form consists of four steps and acquires the most important information for an API proposal. Figure 5.4 shows an example of the *Proposal Creation View*. The first step asks for general information about the API proposal which includes e.g. the API name, the pricing and the purpose description of the API. The second step is considered as optional and can be skipped. This particular step requires the user to provide API-related files for upload. The user has the possibility to upload a Swagger-based[5] API specification or also other files like mock-ups. The third step mainly displays the provided information from the previous two steps for confirmation and enables a submission button for sending the API proposal into the system. Finally, the last step notifies the user about a successful or unsuccessful submission.

[5]https://swagger.io/

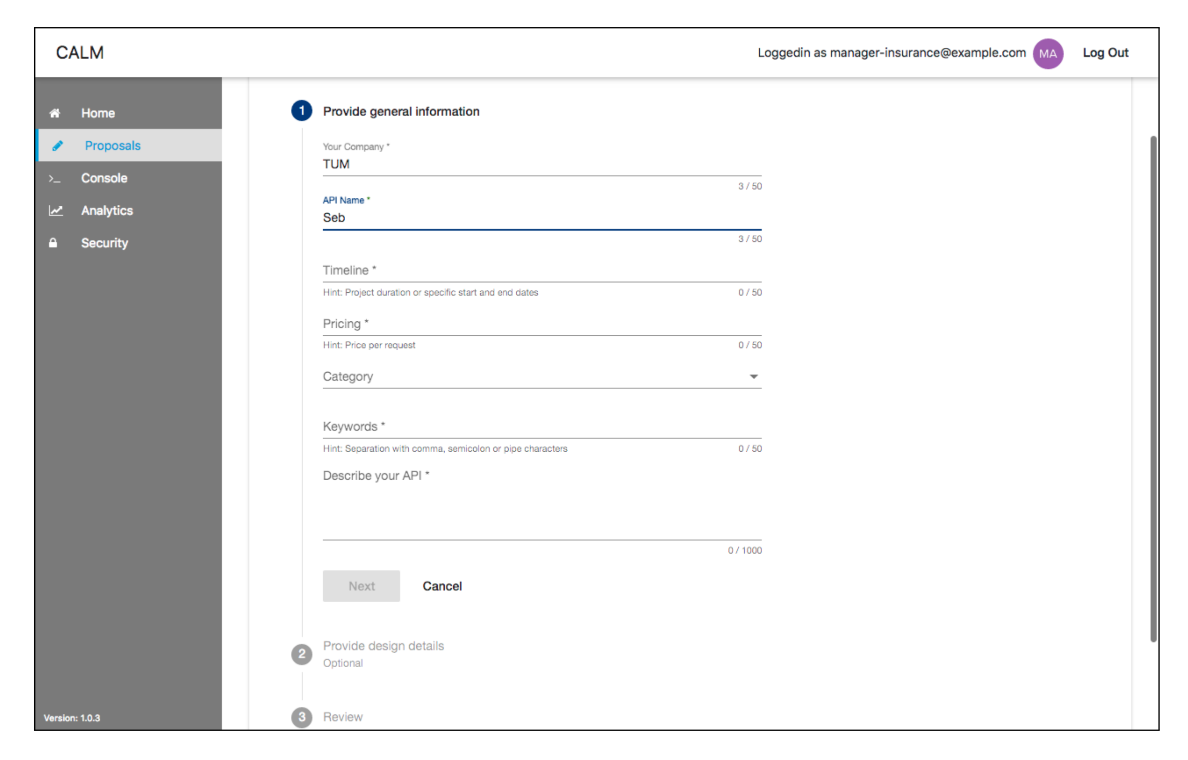
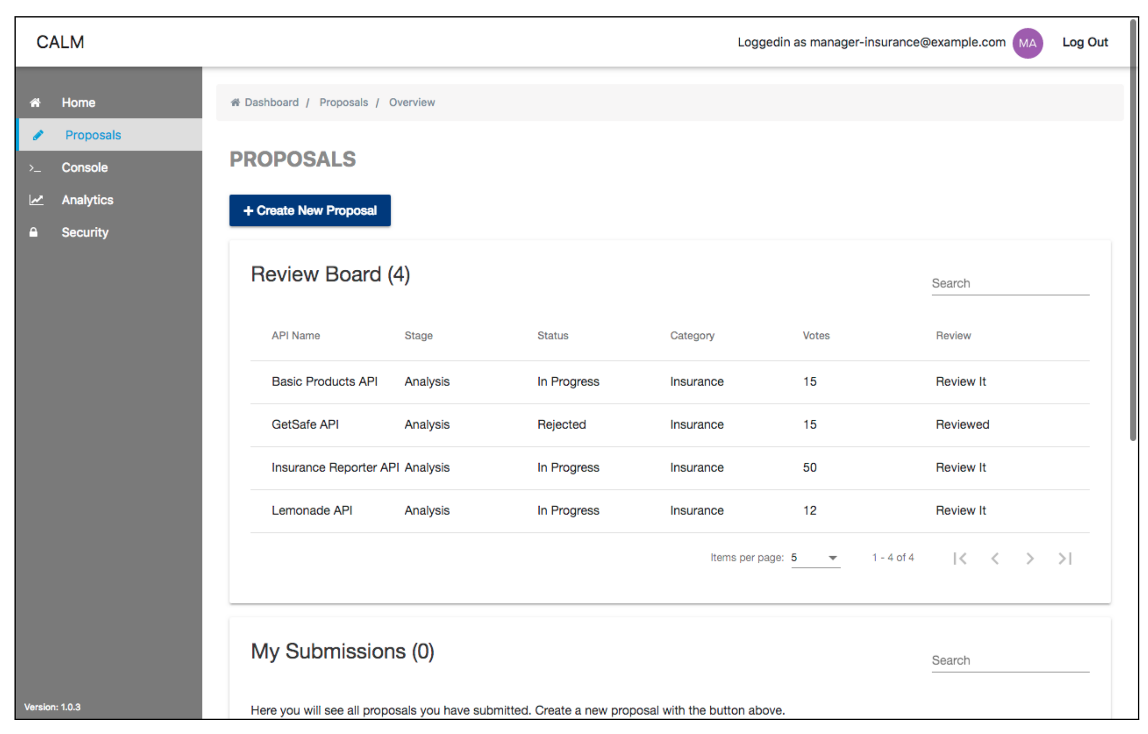


Figure 5.4.: An example of the prototype's *Proposal Creation View*

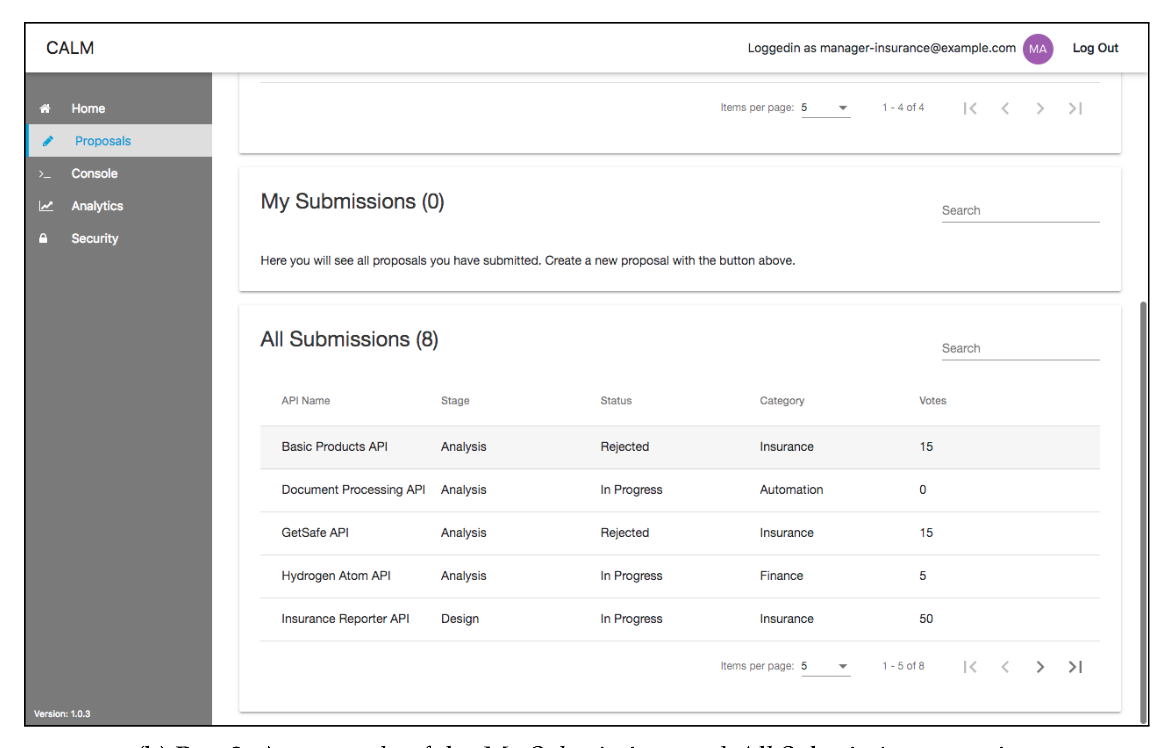
Proposal Overview View

The main focus of the *Proposal Overview View* is to show all API proposals that are persisted in the web application's database. In addition, a further shortcut to the *Proposal Creation View* is provided to the user in form of a blue button at the top of the page. Generally, the page is divided into three containers which classify the API proposals into three predefined groups.

Figure 5.5a shows the *Review Board* container which includes API proposals that need to be reviewed or are already reviewed. While this container is only visible to manager and admin accounts, the two containers in Figure 5.5b are viewable by all authenticated user account types (cf. Table 4.6). The *All Submissions* container holds all API proposals, submitted by all available users of the web application prototype. The *My submissions* container implies only API proposals, submitted by the current authenticated user account. In general, all container boxes include tables to organize the proposal data. All the tables are by default paginated, provide a search text field to filter the data and the columns of the tables can be sorted.



(a) Part 1: An example of the *Review Board* container



(b) Part 2: An example of the *My Submissions* and *All Submissions* container

Figure 5.5.: An example of the prototype's *Proposal Overview View*

Proposal Details View

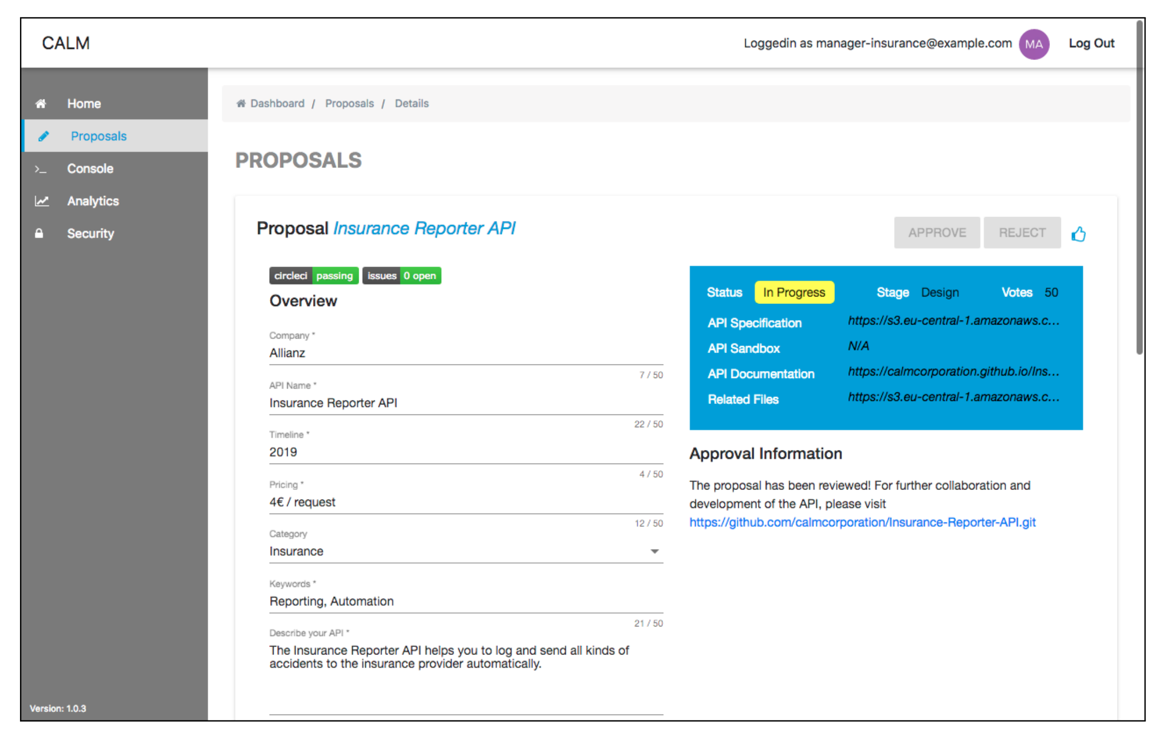
The *Proposal Details View* holds a variety of features. Since an API proposal is considered as the common medium for the collaboration between API consumer and API provider, various collaboration features are provided on that page (cf. section 4.3). Besides, an automated project generation approach for a successfully approved API proposal is also included to accelerate the lifecycle process (cf. section 4.4.2). Figure 5.6 shows an example screenshot of that page.

In general, the *Proposal Details View* gives the user the possibility to vote for an API proposal and to make discussions with other people about the proposal details. Voting is indicated by clicking on the „thumb icon", located at the top right of the API proposal page. Discussions take place in the commenting system at the bottom of the page. Besides text, further materials like files or pictures can be attached into a comment. The commenting system is using a third party service, offered by *Disqus*[6]. Figure 5.6a shows the upper half of the API proposal's page. While the left side gives an overview about the general information of the API proposal like e.g. API name, description, etc., the right side provides the most relevant information of the API proposal in form of a blue rectangle-shaped box. This blue box shows for instance the current lifecycle status and development-related documents of an API proposal. Below the blue box, important approval status information is displayed.

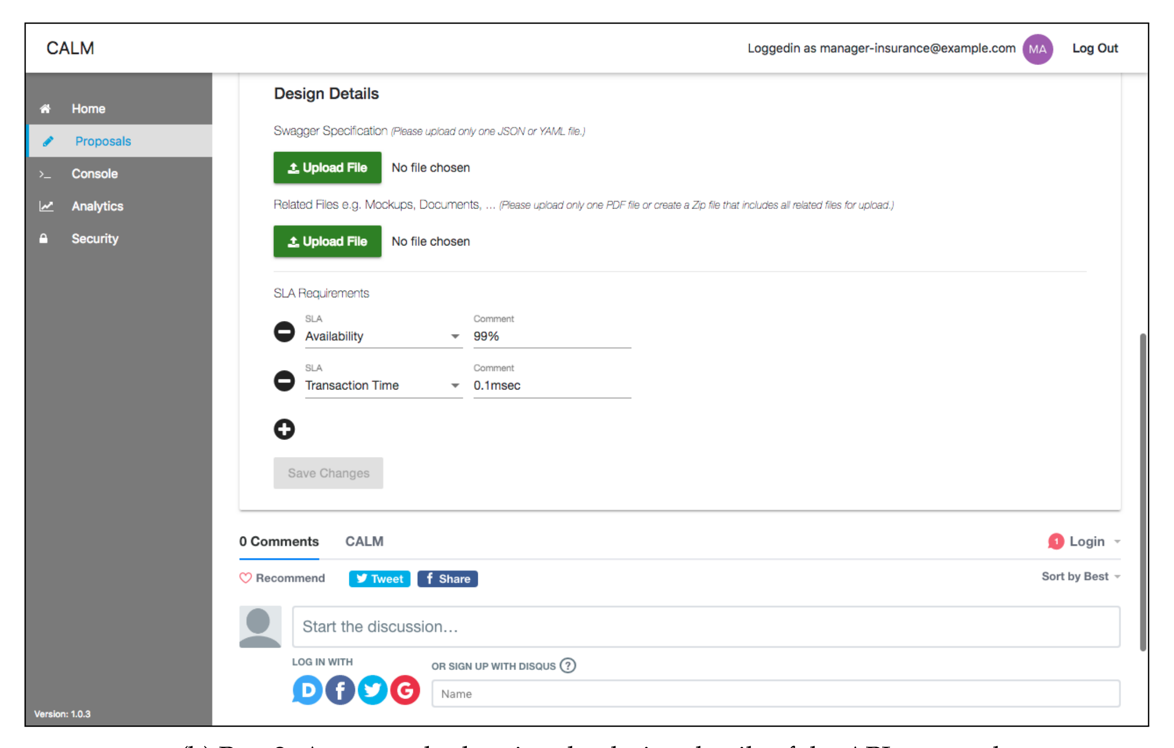
Further design details for an API proposal are also shown in the *Proposal Details View*. Figure 5.6b shows the lower half of the *Proposal Details View*. Like in the *Proposal Creation View*, the lower half offers again the possibility to upload files which includes an API specification or other relevant files. In case, some files were already uploaded during the proposal creation phase, the new uploaded files would overwrite the old ones. Generally, the uploaded files are shown in the blue box on top of the *Proposal Details View* and can be viewed by clicking on the respective hyperlink. With regard to the API specification upload, the UI will notify the user of any structural violations for an uploaded API specification. The violation results are generated with the help of the *API Specification Linter* service (cf. section 5.1). An example output is shown in Figure 5.7.

Another design detail are Service Level Agreements (SLA). The SLAs can be optionally defined and dynamically added to an API proposal by the „plus icon", shown in Figure 5.6b. The idea of SLAs is to make sure that the production-ready version of the API fulfills a certain service quality standard. This is particularly important for API consumers, since the proposed API will be used in the API consumer's own service like in form of a mobile app or website.

[6]https://disqus.com/



(a) Part 1: An example showing the general information about an API proposal



(b) Part 2: An example showing the design details of the API proposal

Figure 5.6.: The Proposal Details View of the web application prototype

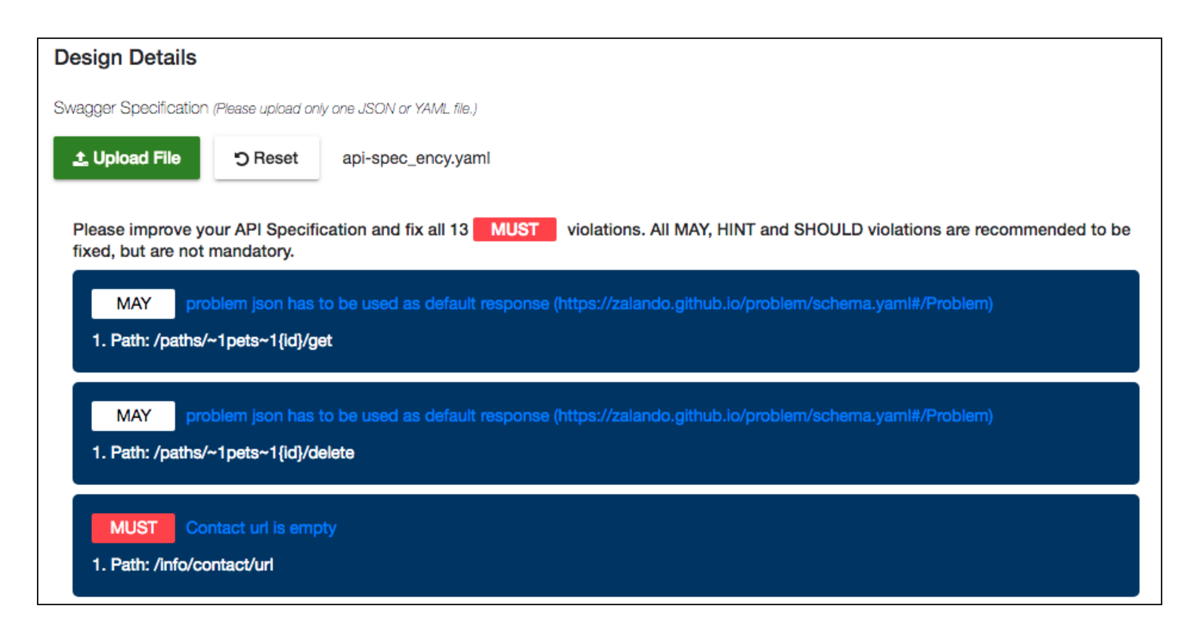
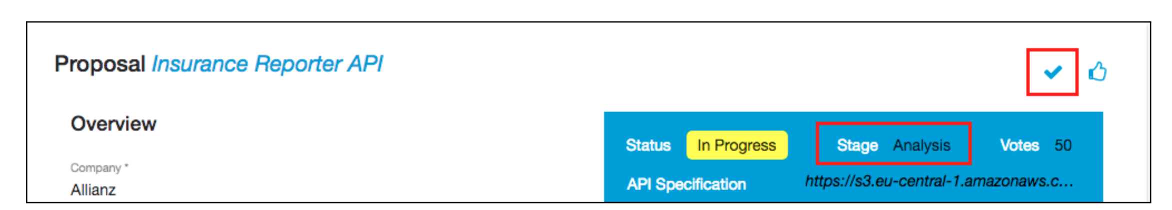


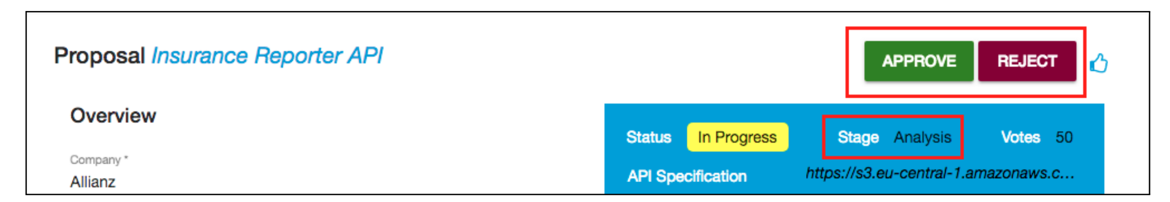
Figure 5.7.: An example output from the API Specification Linter after uploading the Swagger API Specification to the AWS S3 bucket file server

Another aspect to be highlighted in the *Proposal Details View* is the review system for proposals which was elaborated in section 4.4.2. The realization of the review system can be seen in Figure 5.8 and mainly follows the conceptual steps in the state machine diagram, illustrated in Figure 4.5. Once a proposal is submitted by an API consumer, the enterprise architect (API provider) can send an approval request for that API proposal. This is indicated by the „tick icon“ in Figure 5.8a. After sending the approval request, the API proposal switches into the review mode, shown in Figure 5.8b. The enterprise architect can choose to approve or reject a proposal, by clicking the respective button shown in the UI. Assuming that an API specification is already provided and the proposal will be approved, the automated generation of the API development environment takes place which includes the setup of the CI/CD, the Github repository and the API documentation. As already mentioned in the architecture section (cf. section 5.1), the API mock could be implemented yet and is subject to future work. Figure 5.6a and 5.8c shows an example of the UI for an approved API proposal.

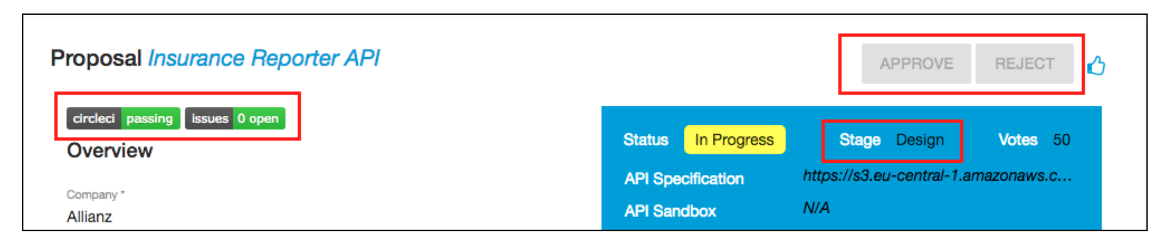


(a) Step 1: Review System - Get approval request for API proposal

Figure 5.8.: The Review System: Transition from sending an approval request to approving an API proposal



(b) Step 2: Review System - Review API proposal



(c) Step 3: Review System - API proposal approved and reviewed

Figure 5.8.: The Review System: Transition from sending an approval request to approving an API proposal (Cont'd)

5.4. Backend

The previous section had a look into the overall front-end structure of the web application prototype. The next section highlights important aspects of the back-end development. The back-end for the prototypical solution consists of three components. The three components are the BFF, the *Proposal Management* service and the *API Specification Linter* service. All three are shortly described in the following.

Backend for Frontend

The main idea of the BFF is to ease the communication between the front-end and various services. The BFF functions as a middle layer and realizes a Façade Design Pattern. The Façade Pattern *„provide[s] a unified interface to a set of interfaces in a subsystem."* (Patni 2017). The main advantage of that pattern is to reduce complexity and to support an easy management of the back-end code base. At the beginning of the implementation phase of this thesis, it was unsure, if only one back-end should be used to handle all the necessary logic or if several smaller services with different technologies would be better. At the end, the decision led to a microservice architecture approach. By the use of the Façade Design Pattern for the BFF, it was then unimportant which technologies each single service would have supported. The main result is that the back-end architecture is flexible, agile and further services can be attached to the *Service Platform* component of the prototype's back-end architecture.

Proposal Management Service

The first task of the *Proposal Management* service is to handle the manipulation of API proposal data. The service provides REST endpoints to retrieve, create, update or delete data. For that, the service is connected to a database which stores that data. The details for the database's data model can be seen in section 4.4.4. The second task of the service is to trigger certain functionalities. One functionality is for example the upload of files into a connected file server. But a more important functionality that needs to be highlighted is the automated generation of the API development environment. For the automation, five initial actions have to be conducted beforehand:

1. Create an Github account
2. Create an organization in that Github account
3. Create several teams inside the organization (1 team = 1 API proposal category)
4. Create a CircleCI account and connect the Github account to it
5. Prepare a template repository in the organization

The general idea of the automation approach is to prepare a template folder structure with all the necessary configuration files inside a repository. Once an approval for an API proposal is triggered in the UI, the template repository is duplicated, adapted to the information of the approved proposal and all the third party development tools are set up. The basis for triggering the development tools is a combination of REST endpoint calls and executing customized script logic. Besides, the API documentation is enabled by using *Widdershins*[7] and *Github Pages*[8] as foundation.

API Specification Linter Service

The *API Specification Linter* service allows to analyze the content structure of API Specification files. Specifically, it focuses only on structural properties, whereas the semantic relevance is ignored. In other words, the service validates the file's syntax, but the service is not able to understand the provided content. In case, the structure of the given Swagger file does not follow predefined rules, a number of violations will be returned as response.

[7] https://github.com/Mermade/widdershins
[8] https://pages.github.com/

The current version of the prototype applies the default rules, defined by *Zalando's RESTful API Guidelines*[9]. However, those guidelines can be adapted with the coding language Kotlin[10]. The adapted guidelines can be then implemented in the open source code of the linter service.

The linting approach in the current prototype requires that firstly, the API Specification file is formatted either as JSON or YAML document and secondly, it is accessible through a URL. In the case of the prototypical solution, the Swagger file is located on an AWS S3 bucket which provides the URL of that file. To lint the API specification, a request with the URL location of the file, as shown in Listing 5.1, need to be sent to the REST endpoint `http://localhost:8080/api-violations`.

```
{
    "api_definition_url": "https://s3.eu-central-1.amazonaws.com/calm-
        bucket/proposals-folder/swagger-api-spec.json"
}
```

Listing 5.1: An example of the request body in JSON format for the API Specification Linter endpoint

After sending the request to the given REST endpoint, a response with various attributes is returned. Listing 5.2 displays an example for a linting output. The output has two major attributes that are relevant for the prototype. The first attribute `violations` holds an array of several violation objects. Each of these objects represents one violation against a specific guideline. The second attribute `violations_count` reflects a statistic and informs about how many violations are represented in each violation category.

Generally, the *API Specification Linter* service makes use of four categories, which are `MUST`, `SHOULD`, `MAY` and `HINT`. The first three are interpreted as described by the RFC2119[11] Standard. The last category HINT is only introduced by Zalando. The `MUST` category can be described as a *required* guideline, the `SHOULD` category as a *recommended* guideline and the `MAY` category as *optional* guideline. The `HINT` category has a minor relevance. The current prototype version pays the highest attention to `MUST` violations and it will not accept any API Specification, if `MUST` violations occur. This means `MUST` violations are required to be fixed, while other violation types can be firstly ignored.

[9]https://opensource.zalando.com/restful-api-guidelines
[10]https://kotlinlang.org/
[11]https://www.ietf.org/rfc/rfc2119.txt

```
{
    ...
    "violations":[
        {
            "title":"Specify Success and Error Responses",
            "description":"operation has to contain the default response",
            "violation_type":"MAY",
            "rule_link":"https://zalando.github.io/restful-api-guidelines/#151",
            "paths":[
                "/paths/~1identifier-types~1{identifier_type}~1source-ids/get"
            ],
            "pointer":"/paths/~1identifier-types~1{identifier_type}~1source-ids/get"
        }
    ],
    "violations_count":{
        "may":1,
        "hint":0,
        "should":0,
        "must":0
    }
}
```

Listing 5.2: An example of the response body in JSON format from the API Specification Linter endpoint

6. Evaluation

After the previously described details of the prototypical implementation for a Collaborative API Lifecycle Management, this coming chapter deals with the evaluation of the prototype to validate assumptions and to gain feedback about the current status of the realized prototypical solution. The evaluation goal is to determine whether the overall concept of the designed solution is suitable in the business environment. Particularly, the evaluation reveals insights about the supporting features for the API Lifecycle and determines, if the chosen features are helpful to overcome the challenges, mentioned in the literature as well as by the industry partner. The evaluation uses the SUS evaluation method from (Brooke 1996) and is extended by further expert interviews in form of open qualitative questions. The structure of the interview setting, the results of the interviews and the possible enhancements for the prototype are presented in the following.

6.1. Expert Interview Setting

Since the evaluation is conducted in form of a case study with expert interviews, the whole interview design follows the recommendations by (Yin 2009). Figure 6.1 depicts the general approach of the interview series to validate the proposed solution in this thesis.

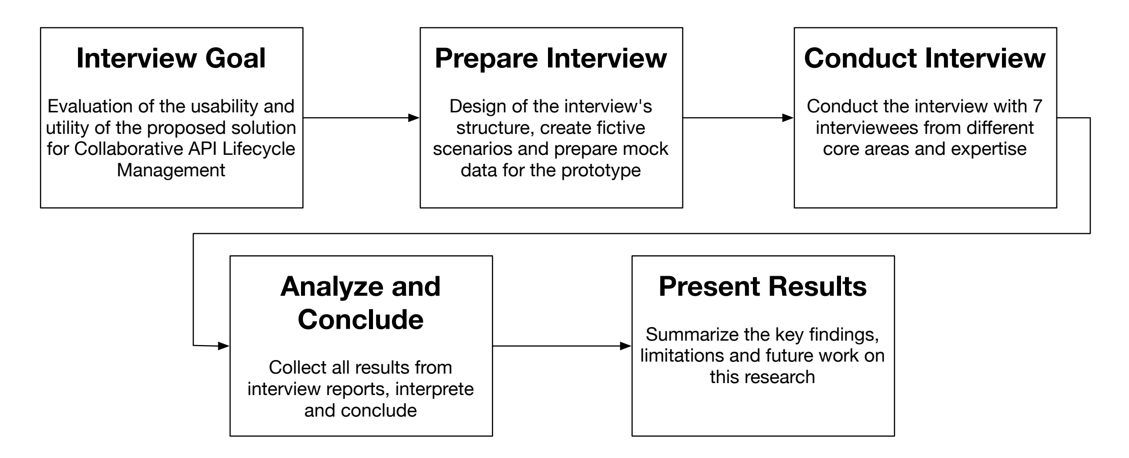


Figure 6.1.: The overall approach of the case study including expert interviews

The evaluation approach begins with the definition of a clear goal for the evaluation process. The goal is set as the *„Evaluation of the usability and utility of the proposed solution for Collaborative API Lifecycle Management"*. The focus lies specifically on the usability and the general utility of the tool within a business environment. The business environment for the evaluation is created through scenarios.

Since there are two main participant groups, which are on the one hand API consumer and on the other hand API provider, two scenarios with typical activities for both groups are designed in the consecutive phase of the interview setting. The details of the interview scenario designs are attached in the appendix B. The first scenario addresses API consumers. The primary goal of the first scenario is to let API consumers submit an API proposal with all its necessary details which includes for instance an API specification or some service level agreements (SLA). The second scenarios focuses on the API provider, especially on the enterprise architect (EA). In the designed API Lifecycle (cf. section 4.2), the EA is the responsible person to review proposals. As shown in the RBAC model (see Table 4.6), an EA, having a manager user role, is only allowed to review proposals from the same domain as his/her own. Thus, the second scenario covers activities such as approving or rejecting a proposal and preparing the proposal for the next lifecycle stage, namely the Dev&Test stage. Each scenario includes actively using the web application prototype to fulfill the activities. After executing each scenario, the interviewees fill out the SUS questionnaire and are asked further open qualitative questions about the designed solution approach.

The interview is conducted with interview partners from different core areas. As shown in Table 6.1, most of the interviewees are enterprise or software architects. The architects are all employees of the industry partner, and all of them have long-time, sounded software development skills. Thus, they are able to undertake both the first scenario for API consumers and the second scenario for API providers. RA1 and RA2 are both research associates from the Technical University of Munich in Germany. Both are not familiar neither with the current nor the target process for APIs in the industry partner's environment. Therefore, both personally undertake only the scenario for API consumers. Nevertheless, the results of the API provider's scenario are shown and explained to them.

The interview setting ends with interpreting the interview results and drawing conclusions. Limitations and further improvements about the current web application prototype are carved out and described in the coming section.

Table 6.1.: The chosen interview partners for the evaluation of the prototype

#	Role	Alias	Years Active	Core Area	Dev Experience
1	Research Associate	RA1	5 years	Knowledge Management, Knowledge Transfer, Software Architecture	7 years
2	Research Associate	RA2	3.5 years	Model based User Interfaces	20 years
3	Enterprise Architect	EA1	6 years	Big Data, Search Technologies, Distributed Systems	20 years
4	Enterprise Architect	EA2	8 years	System Integration, SOA	15 years
5	Software Architect	SW1	3 years	Infrastructure, DevOps	17 years
6	Enterprise Architect	EA3	8 months	Mobile Apps	10 years
7	Enterprise Architect	EA4	3.5 years	Application Integration, SOA	5 years

6.2. Interview Results

In general, the perception of the prototypical solution by the participants is mixed. Beside an overall positive feedback, there are also doubts and concerns about some parts of the proposed solution approach, demonstrated via the web application prototype. The results also show that the current prototypical implementation seems to be better optimized for the API provider side than for the API consumer side. In other words, the features included into the current prototype satisfy the API providers and helps them to overcome typical challenges. However, the API consumer side still lacks of supportive features. Further details about the evaluation outcomes are elaborated in the subsequent sections.

6.2.1. System Usability Scale

The SUS evaluation method is considered as *„valuable evaluation tool, being robust and reliable"* (Brooke 1996). It is a questionnaire with ten standardized statements. Each statement has a 5-point scale, ranging from strongly disagree (1) to strongly agree (5). Before further open questions are asked and discussions are started, SUS is used right after the interviewee has had the chance to use the web application prototype. Also, the interview partners are asked to *„record their immediate response to each item, rather than thinking about items for a long time"* (Brooke 1996). Figure 6.2 shows an example of the SUS questionnaire.

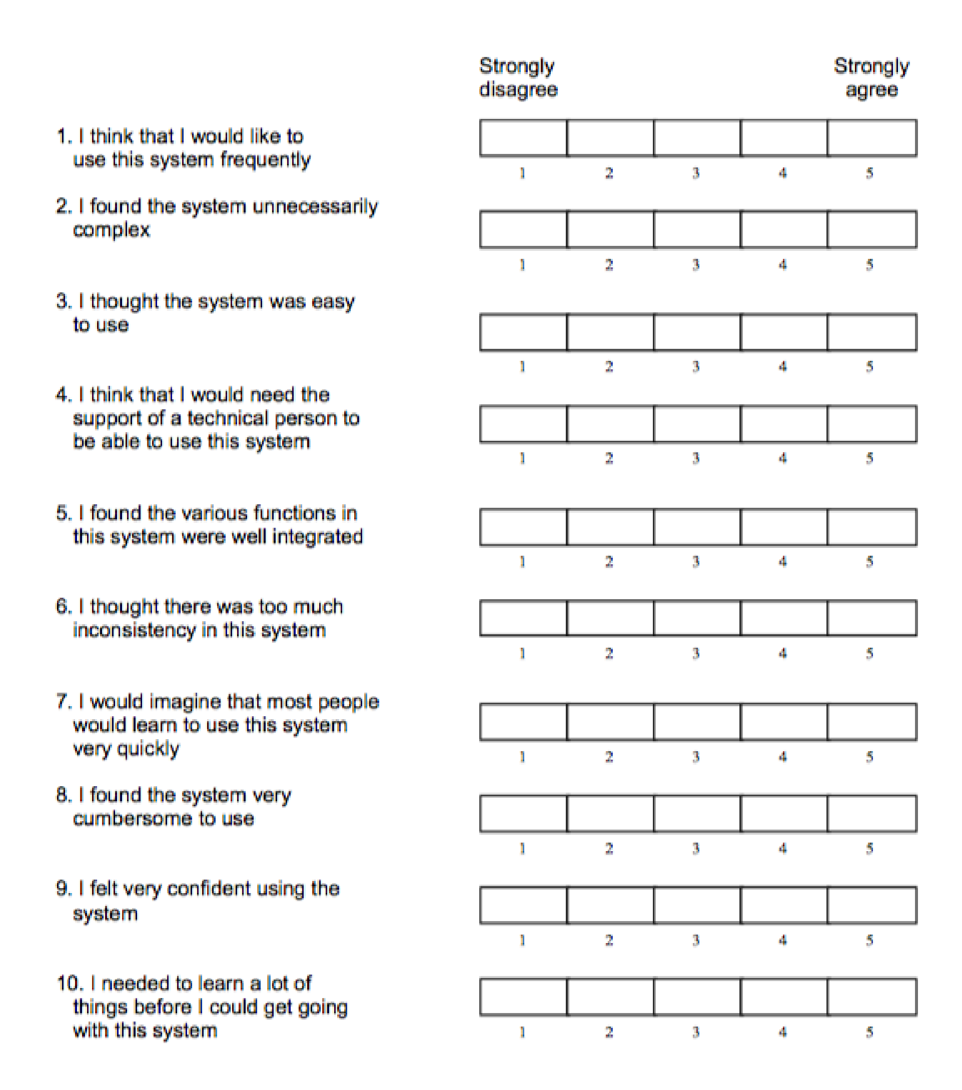


Figure 6.2.: The 10 statements of SUS from (Brooke 1996)

As mentioned in the interview setting in section 6.1, the presented scenarios end each with a SUS questionnaire which each interviewee needs to fill out. The idea of SUS is that participants indicate their personal opinion about the strength of agreement or disagreement for each single statement, with regard to the used web application prototype. The score for each statement has no meaning on its own. As a result, SUS follows a certain calculation. The goal is to have a single number, *„representing a composite measure of the overall usability of the system being studied"* (Brooke 1996). Each statement has a score contribution which ranges from 0 to 4. With regard to the 5-point scale for each statement, the scale position 1 contributes to a score of value 0, and subsequently the next position 2 on the 5-point scale to a score of value 1. Accordingly, this mapping continues until scale position 5, contributing to a score of value 4. The setting of SUS consists of alternating positive and negative statements. Thus, for all statements 1, 3, 5, 7 and 9 the score contribution is calculated as scale position minus 1. Respectively, for all statements 2, 4, 6, 8 and 10, the score contribution results from 5 minus the scale position. Lastly, the scores are summed up and multiplied by 2.5 to achieve the overall SUS score. This calculation mechanism is used for all the interview series. The results for the interview series are summarized in Table 6.2.

Table 6.2.: The SUS score results for scenario 1 (API Consumer) and 2 (API Provider)

#	Alias	Score of Scenario 1	Score of Scenario 2
1	RA1	77.5	-
2	RA2	35	-
3	EA1	77.5	87.5
4	EA2	87.5	95
5	SW1	90	77.5
6	EA3	95	92.5
7	EA4	85	87.5
Average		**78.21**	**88.0**

Generally, the SUS score ranges from 0 to 100. It is hard to understand what an individual SUS score means. Therefore, Bangor et al. 2009 suggest to add adjective ratings, acceptability scores or school grading scales to interpret the individual SUS scores easier. As shown in Figure 6.3, any SUS score above 70 is considered as acceptable. Hence, both average scores for scenario 1 and 2 reveal that the current prototype is usability wise accepted by the majority of the participating interview partners. Nevertheless, the average SUS scores for scenario 1 and 2 also show that the current solution has still space for improvement, especially with regard to the API consumer side.

Particularly, the scores of RA1 and EA1 reflect a C grade, and the score of RA2 even an F grade. Overall, it can be seen, that the support for API consumer and API provider by the current prototype, are experienced differently. Among all the architects (EA1, EA2,

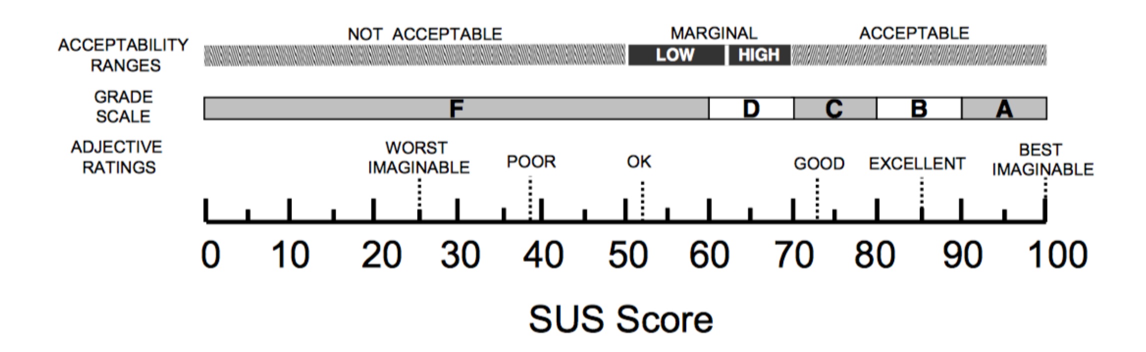


Figure 6.3.: The relationship between the SUS score and adjective ratings, acceptability scores, and school grading scales (Bangor et al. 2009)

EA3, EA4, SW1), there are people favoring one side over the other. Nonetheless, there is not a clear trend towards one direction. SUS generally does not provide insights into each individual interview partner's opinion. As a result, additional open qualitative questions are asked to find out details about the interviewees' assessments. The focus lies particularly on retrieving limitations of the overall solution approach and also recommendations to improve the artifact in further design iterations.

6.2.2. Open Qualitative Questions

Before the end of each interview, the interviewees are asked four open qualitative questions, in order to understand their personal point of view about the current status of the solution design. The four qualitative questions are part of the evaluation script which is shown in appendix B.

In general, the overall judgment about the web application prototype is perceived positive by the majority of the participating respondents. Most of the people like the concept of the proposed solution to support the lifecycle for APIs in form of a web application. Some points about the design artifact that the respondents highlight are:

- API Specification linter to pre-check violations in the provided Swagger/OpenAPI file
- Automated generation of the development setup for an API proposal
- Social features like voting and commenting
- Integration of existing development tools like Github

- Clear, structured and simple UI design

Feedback to Acceleration Approach

In terms of accelerating the API Lifecycle, there is a clear tendency that all architects (EA1, EA2, EA3, EA4, SW1) agree that the current prototype would generally help to make the API Lifecycle faster. Some arguments that are provided by the architects and support this assessment are:

- Transparency over the whole API Lifecycle
- Central place to start the process for getting an API
- Status tracking of the API progress
- Standardized, structured and unified form for acquiring information
- All important information related to an API proposal visible at one place
- API First approach itself increases acceleration
- API stubbing or API mock idea provides independence among API consumer and API provider

In comparison, RA1 and RA2 are either unsure or doubt the acceleration boost. Both are generally skeptical about the API-First approach which is used in the proposed solution design. RA2 adds that he would like to have more user centered design features for the API consumer side. RA2 and EA7 mention also that the current prototype might be only optimized for the API provider side. As already seen in a previous section, this argument is also reflected in the average SUS scores, shown in Table 6.2.

Feedback to Collaboration Support

Regarding the collaboration among the different stakeholders within the Collaborative API Lifecycle, the opinions among the interviewees widely differ. In general, slightly more of the participating interview partners express a positive point of view. The results do not show a clear tendency towards one direction. It slides between a rather neutral and a positive point of view. The included collaboration features like comments and votes add higher collaboration value. Comments are useful for discussions which are directly

visible below the API proposal details. Votes are also useful to filter popular proposals, driven by the market need. Nonetheless, there could be further features added to achieve even higher collaboration among the participants. Further details are elaborated in the next section 6.3. According to EA2 and EA3, collaboration value is only achieved, if all the participants are actively using the provided features. Without any active commitment, both will doubt that the collaboration features will create any value. EA3 even adds that people might misuse the comment feature to negotiate the price of a proposal. In terms of communication, he further extends his point of view by mentioning that comments might be more helpful for external customers rather than for internal employees.

6.3. Concerns and Possible Enhancements of the Solution Design

The interview results reveal a few limitations and enhancement ideas. Besides technical feedback, some interview partners provide further thoughts for redesigning parts of the general conceptual solution approach. The following section starts with outlining general feedback about the whole solution design. Afterwards, enhancement suggestions are presented individually for each major view of the web application prototype.

6.3.1. General Technical and Conceptual Feedback

Based on the consolidated interview results, the following list presents a set of major technical and conceptual enhancement suggestions for the overall prototype:

Staging model based on API type A few of the interviewed enterprise architects propose to dynamically adapt the staging model of the API Lifecycle in the prototype. This means that the staging model should depend on the chosen API type (Public API, Partner API, Private API) of a proposal. Based on the API type, the length and quality gates of the staging model would be adjusted respectively. Furthermore, it is suggested to show the whole API Lifecycle with the stages inside an API proposal. In addition, the latest stage should be visually highlighted.

Flexible Review System The current design of the prototype does not allow a flexible review system. In particular, once a proposal is rejected, it cannot return to the approval process anymore (see Figure 4.5). However, the prototype should provide the chance to improve the rejected proposal. A possible way could be to collect a list of issues for rejection in form of a checklist and gradually resolve the issues. After all resolved issues, the rejected proposal could restart the review process again.

Further categories for functionally disjunctive APIs One interviewed enterprise architect recommends to extend the classification of API proposals. Beside the domain or category of a API proposal, the enterprise architect wishes for further granular differentiation. He mentions that the functional coverage of APIs might overlap with each other. To make APIs nearly functional disjunctive will be challenging.

Include further tools Several respondents comment that they prefer the way, how tools like Github or others are currently integrated into the prototype. Generally, it is advised to keep the prototype as supporting tool, like it is now. As a result, the integration of new tools should be always evaluated for suitability. The majority also adds that the complete integration of other technical tools should never be the goal. The UI would be otherwise overloaded and the usability would decrease. However, an enterprise architect names Atlassian Jira[1] and Atlassian Confluence[2] as further considerable tools.

Provide more examples, context information and constraints beforehand Generally, several respondents suggest that the users should be given more context information to guide them along the API Lifecycle process. They should be always kept updated about subsequent steps or even direct them to the following step automatically. Furthermore, examples and constraints should be provided to the user beforehand. This particularly helpful, when a user creates the API Specification.

6.3.2. Feedback for the Home View

Based on the consolidated interview results, the following list presents a set of major technical enhancement suggestions for the *Home View*:

More powerful search or filter element The majority of the respondents name the *Category Filter* element as insufficient. Most of the respondents would like to have a faceted search or even just a search text field. By using a faceted search, the user would have predefined criteria that he could choose from. A search text field would provide more flexibility. A combination of both UI elements might be also imaginable. Nevertheless, there is no common agreement among the participating interview partners, what type of search is suited best for the UI.

Add Gamification to support search Several interviewees mention that the API consumer might not even want to search for existing proposals. Instead, the API consumer would just directly create a new API proposal. One respondent suggests to add Gamification to motivate users to search for a proposal.

[1]https://www.atlassian.com/software/jira
[2]https://www.atlassian.com/software/confluence

6.3.3. Feedback for the Proposal Creation View

Based on the consolidated interview results, the following list presents a set of major technical enhancement suggestions for the *Proposal Creation View*:

Set API type and proposal visibility Some interview partners propose to allow the user to set the type of the API, whether it is a internal, partner or external API. Beyond that, some also recommend not to set all proposals to public by default. On API consumer side, there might be some cases, where APIs should be only visible to a limited number of people.

Use of date picker for availability date of API For most of the interviewees, the length of the development period for the API is uninteresting. Instead, it is suggested to include only the end date, when the API should be available for production usage. Also, a date picker UI element would be more helpful for the user to easily select the date visually.

Save unfinished form state Some interviewees ask for storing the states of unfinished forms. Firstly, if a user accidentally navigates to other pages, he or she should get a notification alert. Secondly, the state of the inserted information into an unfinished form should be restorable. The current version of the prototype would for instance erase all inserted information in the *Proposal Creation View*, if the user has not submitted the proposal yet and switches to another page.

6.3.4. Feedback for the Proposal Overview View

Based on the consolidated interview results, the following list presents a set of major technical enhancement suggestions for the *Proposal Overview View*:

Introduce different voting level thresholds In order to visually catch the high demanding API proposals easily, one interviewee recommends to classify and visually color proposals, based on the number of their votes. Thresholds could be introduced, which means that e.g. proposals with 0-10 votes are colored red, proposals with 10-100 votes are colored yellow and proposals with votes above 100 are colored green.

Action shortcuts for reviewing proposals An action shortcut for voting was already considered during the mockups creation phase for the design artifact. Due to the chosen UI framework, this feature could not be implemented in the current prototype. Nevertheless, it could be included in the next design iteration and extended by an action shortcut for accepting or rejecting a proposal.

Provide new container for reviewed proposals The *Review Board* container holds, on the one hand, proposals that need to be reviewed and on the other hand, proposals that has been already reviewed. One respondent recommends to separate them into two different containers, because he perceives that the *Review Board* container should contain only proposals that still need to be reviewed. Additionally, both kinds of proposals should be visually distinguishable.

6.3.5. Feedback for the Proposal Details View

Based on the consolidated interview results, the following list presents a set of major technical enhancement suggestions for the *Proposal Details View*:

More status information from integrated tools Some respondents suggest to include more information from integrated tools like those about the development status from Github, since the development status is closely related to the overall status of an API. The included information could be issues, discussions, version numbers, etc. Moreover, the prototype is used by a variety of people with different roles. Hence, most of the displayed information should be understandable for both with and without intense IT knowledge.

Github as main source of information Another possibility is to use Github as main source of information or in other words, all information, related to an API proposal, should be stored in the repository. The necessary proposal information in the the web application prototype is then retrieved from the repository. In this case, the API related files could be just linked to the location in the repository which would replace the file upload features. The NPM website[3] is mentioned as an example by an interviewee. NPM package pages use the same described approach and retrieve the information to be displayed from the respective Github repository.

More tool support for API Consumer The additional tool support for API consumers is perceived by the respondents differently. Nevertheless, some would like to have a data model and an SDK provided by the API provider, in order to create the API specification easier. Additionally, a visual tool support for the creation of the API specification is also mentioned. One respondent mentions *Restlet*[4] as example.

History Record Feature One important collaboration feature that is recommended by several interview partners is a history record feature. The idea of history record feature is to record all changes that have been made to a proposal e.g. in form of a list or a timeline. In addition, some also suggest to record the reason for proposal review decision.

[3]https://www.npmjs.com/

[4]https://restlet.com/modules/studio/

Annotation Feature Some interviewees wish for a way to annotate an API proposal with graphics like diagrams or pictures to support the discussion point. In the current prototype, graphics can be attached as files in the discussion section. Nonetheless, an extra feature is considered more suitable.

Include quality gates for switching stages Some interviewed architects asked for including the quality gates from the conceptual model directly into the prototype. The quality gates could be visualized as a list of checkpoints. If all the checkpoints are fulfilled, the stage of the API proposal will change.

Progress Tracking Feature Another feature, related to the status of an API, is a progress tracking feature. The progress of an API proposal could be shown as a progress bar. The fill level of the progress bar would then depend on, either how many quality gates are fulfilled, or how many issues are solved on Github. A combination of both approaches would be possible too.

Adapt representation of API Specification Linter results Firstly, a few interview partners recommend to provide hints about the linting process to the user beforehand, since the pop-up of the linting results in the current version of the prototype is considered as too surprising to the user. Secondly, the visual display of the API Specification linting results should be adapted. A traffic light or other color codes could be introduced to differentiate the level of violations.

Consider Ratings instead of Votings A respondent suggests to use ratings instead of votings. Unfortunately, a clear reason is not mentioned by the interviewee. During the conception of the prototype, a rating system was also considered. The difference between a rating system and a voting system is that a rating system expresses the value of an API proposal, whereas a voting system the importance of an API proposal on the market. In general, both are suitable and express similar intentions. Ideally, the second research iteration would test the two features in a A/B testing environment.

Save changes automatically and manually There is no common agreement among the participating interview partners, whether changes on an API proposal should be saved automatically or manually by clicking a button. The current prototype requires a manual approach. Generally, both approaches are considered suitable. One interviewee even wishes for a combination of both ways. He recommends for file uploads a manual saving approach due to confirmation reasons, whereas for all textual details or SLAs an automatic saving would just be enough. However, further validation with A/B testing is needed.

6.4. Synthesis of Evaluation Results

The evaluation chapter provides insights about the perception of the prototype for Collaborative API Lifecycle Management. For that, a case study is conducted in form of expert interviews. Section 6.1 shows the setting of the evaluation approach and the chosen interview partners. Generally, the majority of the participating interview partners state that the overall solution design approach is viable, adds value, and supports both API consumer and API provider through the API Lifecycle process.

On the one hand, the evaluation results reveal that the prototype clearly accelerates the lifecycle process for APIs. Based on the results from the SUS scores and from the open qualitative questions, it can be seen that the interview partners are divided into two parties. All interviewed architects show a positive support for the API First approach, while the interviewed research associates, reflecting the external API consumers, doubt it. Further details are elaborated in section 6.2.2.

On the other hand, the evaluation results show that the included collaboration features like comments and votes are beneficial. However, some architects added that the collaboration features need generally active commitment by all users, in order to reach their full potential. Section 6.2.2 provides further details.

Since the evaluation of the design artifact is only conducted once in this thesis, there is still much space for improvement in the solution design approach. The participating respondents suggest a list of several technical and conceptual enhancement ideas that can be included in subsequent validation iterations. Table 6.3 summarizes the provided feedback and references the details of each feedback to the respective section in this thesis.

Table 6.3.: A summary of the current prototype's feedback acquired by expert interviews

	Feedback Title	**Section**
	General Feedback	
01	Staging model based on API type	6.3.1
02	Flexible Review System	6.3.1
03	Further categories for functionally disjunctive APIs	6.3.1
04	Include further tools	6.3.1
05	Provide more examples, context information and constraints beforehand	6.3.1
	Feedback for Home View	
06	More powerful search or filter element	6.3.2
07	Add Gamification to support search	6.3.2
	Feedback for Proposal Creation View	
08	Set API type and proposal visibility	6.3.3
09	Use of date picker for availability date of API	6.3.3
10	Save unfinished form state	6.3.3
	Feedback for Proposal Overview View	
11	Introduce different voting level thresholds	6.3.4
12	Action shortcuts for reviewing proposals	6.3.4
13	Provide new container for reviewed proposals	6.3.4
	Feedback for Proposal Details View	
14	More status information from integrated tools	6.3.5
15	Github as main source of information	6.3.5
16	More tool support for API Consumer	6.3.5
17	History Record Feature	6.3.5
18	Annotation Feature	6.3.5
19	Include quality gates for switching stages	6.3.5
20	Progress Tracking Feature	6.3.5
21	Adapt representation of API Specification Linter results	6.3.5
22	Consider Ratings instead of Votings	6.3.5
23	Save changes automatically and manually	6.3.5

7. Conclusion

The following chapter provides a conclusion about the research results, gained in this thesis. First, the key findings for each RQ are summarized. Furthermore, possible limitations for this research are presented. Finally, a short insight into possible future work is given.

7.1. Summary

The coming section summarizes the key findings for each individual RQ.

RQ1: How could a holistic approach for an API Lifecycle, including phases, activities, artifacts and roles, look like that is driven by the collaboration of participating stakeholders?

Before building the conceptual model for CALM, it was necessary to define the requirements first. The requirements were retrieved from literature and conducted expert interviews. The results are summarized in section 4.1. In addition, existing API Lifecycles and similar lifecycle approaches were compared and presented in section 2.3.3. Based on the gained knowledge, the CALM model was formed with the help of further literature and expert interviews. The current conceptual model is optimized for the industry partner's target API Lifecycle process. Compared to similar lifecycle approaches, the resulted lifecycle model for APIs has both a product and service character. In addition, the API Lifecycle model is flexible, agile and supports continuous improvement. The resulted conceptual model for CALM is elaborated in section 4.2. Generally, the CALM model covered all major challenges from the industry partner shown in section 4.1.1. Further requirements from literature, shown in section 4.1.2, were also fully covered by the Full Lifecycle API Management model which includes the CALM model (cf. Figure 4.1).

RQ2: How can tools and collaborative features be used to support the API Lifecycle Management?

To assist the conceptual model, a prototypical implementation as design artifact was conceived and developed. The design artifact has both a collaboration and acceleration

support. In general, the collaboration between API consumer and provider is based on an API proposal. The chosen collaboration features include a voting system and a commenting system which are attached to each API proposal (cf. section 4.3). To advance the lifecycle process, an acceleration approach including an automated development environment generation was added to the prototype. The planned API mock feature to make API provider and API consumer independent could not be realized in the current version of the design artifact yet. The system design of the prototype is described in section 4.4 and the implementation details are elaborated in chapter 5.

RQ3: What are the users' experiences of the designed web application prototype solution?

The evaluation of the prototype was conducted in form of a case study with expert interviews. The expert interviews were conducted with chosen participants from the industry partner as well as researchers from the Technical University of Munich. The evaluation goal was to assess the usability and utility of the proposed solution for Collaborative API Lifecycle Management. Two scenarios with typical tasks for API consumer and provider were created that the interviewees had to follow during the evaluation process. The usability assessment was conducted by using the SUS evaluation method. Additionally, further open qualitative questions were asked.

The evaluation results showed that the prototype is a viable solution. There was a clear tendency towards a positive feedback for the acceleration approach. The acceleration through a structured form and the automation approach were considered suitable. Nevertheless, the results showed that the architects from the industry partner tended to support the API First approach, while the interviewed researchers, representing external developers, were skeptical towards this paradigm. In terms of collaboration, there was just a slightly more positive feedback. A bit less than the half of the interviewees kept a neutral position. The results showed that collaboration generally needs active commitment, otherwise the existing collaboration features will not be used.

Overall, all the general idea of the proposed solution was considered helpful for both API consumer and provider. The chosen collaboration features like the commenting system and voting system were perceived positively. Further highlights of the prototypical implementation include the API specification linter and the integration of existing tools like Github. Details to the evaluation approach and results are described in chapter 6.

7.2. Limitations

In general, there are a few known limitations that got an impact on the research results of this thesis. The following section outlines the major limitations.

In general, this thesis applied the DSR approach by (Hevner et al. 2004). Due to the narrow time frame of this master's thesis, the assessment-and-refinement cycle between building and validating the design artifact could be only conducted once. Particularly, weaknesses of the first version of the prototype could be identified, but the prototype has not been improved yet.

Another limitation is that the whole research was conducted in only one company and the majority of the interviewed people were employees from the industry partner. A bigger number and a wider diversity of interview partners would have strengthened the validity of the interview results.

With regard to the interview setting, the interview procedure was subject to natural cognitive biases, impacting the evaluation results. Since there were two scenarios for the architects to be executed with the prototype, the impressions of the first scenario might have influenced the impressions of the second one. In other words, the SUS score from scenario 1 and 2 were affected by each other. This phenomenon is also known as *Priming*. Further psychological effects might have occurred and need to be considered.

Lastly, further API protocols like GraphQL could have been considered for research besides REST. This would have included some technology adaptions and might have impacted the overall solution design approach.

7.3. Future Work

The key findings of this thesis can be used as basis for various future research. Some insights into potential future work is given in the following.

The previously described section gave an overview about known limitations of this thesis. Naturally, all of them could be possibly addressed in future research. With regard to limitations, this thesis provided also a summary of further technical and conceptual enhancement ideas for the design artifact that was proposed by the interview partners. A summary of the interview feedback is shown in the Table 6.3. Overall, the evaluation results revealed that in terms of usability, the prototype still left space for improvement.

Particularly, the API consumer side need to be given more attention towards user-centered design. Additional methods like Gamification or Machine Learning might ease the experience when using the prototypical solution.

Moreover, the current prototype for Collaborative API Lifecycle Management realized only a limited number of all the use cases, shown in Figure 4.3. The prototypical solution could be extended by the remaining use cases or further ones could be carved out, using the results of this thesis as basis.

Last but not least, the conceptual model for API Lifecycle Management was formed using literature foundations and the inputs from expert interviews with the industry partner. In order to create a generic lifecycle model for APIs, further validations with other companies need to be conducted.

To conclude, the prototype is overall a viable solution that accelerates the API Lifecycle Management and offers helpful collaboration features. It provides value to both API consumers and API providers.

Bibliography

Apigee (2016a). *API Best Practices - Managing the API Lifecycle: Design, Delivery, and Everything In Between*. Ebook. URL: https://pages.apigee.com/rs/351-WXY-166/images/API-Best-Practices-ebook-2016-12.pdf (accessed on Mar. 13, 2018).

Apigee (2016b). *The State of APIs - 2016 Report on Impact of APIs on Digital Business*. Report. Apigee. URL: https://pages.apigee.com/ebook-State-of-APIs-reg.html (accessed on Mar. 9, 2018).

Bangor, Aaron, Philip Kortum, and James Miller (2009). "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale." In: *J. Usability Studies* 4.3, pp. 114–123.

Bermbach, David and Erik Wittern (2016). "Benchmarking Web API Quality." In: *Web Engineering*. Ed. by Alessandro Bozzon, Philippe Cudre-Maroux, and Cesare Pautasso. Vol. 9671. Cham: Springer International Publishing, pp. 188–206. DOI: 10.1007/978-3-319-38791-8_11. (Accessed on Aug. 16, 2018).

Bloch, Joshua (2006). "How to Design a Good API and Why It Matters." In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*. OOPSLA '06. New York, NY, USA: ACM, pp. 506–507. DOI: 10.1145/1176617.1176622.

Brodsky, Laura and Liz Oakes (2017). *Data sharing and open banking*. Data sharing and open banking. URL: https://www.mckinsey.com/industries/financial-services/our-insights/data-sharing-and-open-banking (accessed on Apr. 4, 2018).

Brooke, John (1996). *SUS: A quick and dirty usability scale*.

Brooke, John (2013). "SUS: A Retrospective." In: *J. Usability Studies* 8.2, pp. 29–40.

Bruegge, Bernd and Allen H. Dutoit (2010). *Object-oriented software engineering: using UML, patterns, and Java*. 3rd ed. Boston: Prentice Hall. 778 pp.

CA Technologies (2015). *The API Management Playbook*. URL: https://www.ca.com/content/dam/ca/us/files/ebook/the-api-management-playbook.pdf (accessed on Mar. 18, 2018).

Chow, Tsun and Dac-Buu Cao (2008). "A survey study of critical success factors in agile software projects." In: *Journal of Systems and Software* 81.6, pp. 961–971. DOI: 10.1016/j.jss.2007.08.020. (Accessed on Mar. 20, 2018).

Collins, George and David Sisk (2015). *API economy - From systems to business services*. Extract from "Tech Trends 2015" p. 23-33. Deloitte University Press. URL: https://www2.deloitte.com/content/dam/insights/us/articles/tech-

trends-2015-what-is-api-economy/Tech-Trends-2015-FINAL_3.25.pdf (accessed on Mar. 21, 2018).

Cooper, Robert G. and Elko J. Kleinschmidt (1995). “Benchmarking the Firm’s Critical Success Factors in New Product Development.” In: *Journal of Product Innovation Management* 12.5, pp. 374–391. DOI: 10.1111/1540-5885.1250374.

Daimler (2018). *Mercedes Benz API Portal*. URL: https://developer.mercedes-benz.com/ (accessed on Sept. 24, 2018).

De, Brajesh (2017). *API Management*. Berkeley, CA: Apress. DOI: 10.1007/978-1-4842-1305-6.

Earls, R. Alan (2013). *Digging into quality: API best practices, problems and advice*. Why use new lifecycle tools in API management platforms? URL: https://searchmicroservices.techtarget.com/feature/Digging-into-quality-API-best-practices-problems-and-advice (accessed on Aug. 24, 2018).

Evans, Peter C. and Rahul C. Basole (2016). “Revealing the API ecosystem and enterprise strategy via visual analytics.” In: *Communications of the ACM* 59.2, pp. 26–28. DOI: 10.1145/2856447.

Fagerholm, Fabian and Jurgen Munch (2012). “Developer experience: Concept and definition.” In: IEEE, pp. 73–77. DOI: 10.1109/ICSSP.2012.6225984.

Fielding, Roy Thomas (2000). “Architectural Styles and the Design of Network-based Software Architectures.” PhD thesis. University of California, Irvine.

Fischbach, Michael, Thomas Puschmann, and Rainer Alt (2013). “Service Lifecycle Management.” In: *Business & Information Systems Engineering* 5.1, pp. 45–49. DOI: 10.1007/s12599-012-0241-5.

Fowler, Martin (2010). *Richardson Maturity Model*. Richardson Maturity Model. URL: https://martinfowler.com/articles/richardsonMaturityModel.html.

Fremantle, Paul, Jacek Kopecky, and Benjamin Aziz (2015). “Web API Management Meets the Internet of Things.” In: DOI: 10.13140/rg.2.1.3133.5840.

Gemechu, Eskinder Demisse, Guido Sonnemann, Arne Remmen, Jeppe Frydendal, and Allan Astrup Jensen (2015). “How to Implement Life Cycle Management in Business?” In: *Life Cycle Management*. Ed. by Guido Sonnemann and Manuele Margni. Dordrecht: Springer Netherlands, pp. 35–50. DOI: 10.1007/978-94-017-7221-1_4.

Giebel, M., H. Essmann, N. Du Preez, and R. Jochem (2009). “Improved innovation through the integration of Quality Gates into the Enterprise and Product Lifecycle Roadmaps.” In: *CIRP Journal of Manufacturing Science and Technology* 1.3, pp. 199–205. DOI: 10.1016/j.cirpj.2008.10.004.

González, Francisco Javier Miranda and Tomás Manuel Bañegil Palacios (2002). “The effect of new product development techniques on new product success in Spanish firms.” In: *Industrial Marketing Management* 31.3, pp. 261–271. DOI: 10.1016/S0019-8501(00)00150-4.

Google (2018). *Apigee API Management*. URL: https://apigee.com (accessed on Sept. 24, 2018).

Haupt, Florian, Frank Leymann, and Karolina Vukojevic-Haupt (2017). "API governance support through the structural analysis of REST APIs." In: *Computer Science - Research and Development*. DOI: 10.1007/s00450-017-0384-1.

Hevner, March, Park, and Ram (2004). "Design Science in Information Systems Research." In: *MIS Quarterly* 28.1, p. 75. DOI: 10.2307/25148625.

Holly, Kerrie et al. (2014). *The Power of the API Economy business performance*. IBM Redbooks. URL: http://www.redbooks.ibm.com/redpapers/pdfs/redp5096.pdf (accessed on Mar. 1, 2018).

Iyengar, Keerthi, Somesh Khanna, Srinivas Ramadath, and Daniel Stephens (2017). *What it really takes to capture the value of APIs*. What it really takes to capture the value of APIs. URL: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/what-it-really-takes-to-capture-the-value-of-apis#0 (accessed on Mar. 21, 2018).

Iyer, Bala and Mohan Subramaniam (2015a). *Are You Using APIs to Gain Competitive Advantage?* Are You Using APIs to Gain Competitive Advantage? URL: https://hbr.org/2015/04/are-you-using-apis-to-gain-competitive-advantage (accessed on Mar. 21, 2018).

Iyer, Bala and Mohan Subramaniam (2015b). *The Strategic Value of APIs*. The Strategic Value of APIs. URL: https://hbr.org/2015/01/the-strategic-value-of-apis (accessed on Mar. 21, 2018).

Jacobson, Daniel, Greg Brail, and Dan Woods (2011). *APIs: A Strategy Guide*. O'Reilly Media, Inc.

Jayathilaka, Hiranya, Chandra Krintz, and Rich Wolski (2015). "EAGER: Deployment-Time API Governance for Modern PaaS Clouds." In: IEEE, pp. 275–278. DOI: 10.1109/IC2E.2015.69.

Kepes, Ben (2014). *Software may be eating the world but APIs are giving it teeth*. Diversity Limited. URL: http://www.diversity.net.nz/wp-content/uploads/2014/04/REP-Software-will-Eat-101096-1.pdf (accessed on Mar. 21, 2018).

Kohlborn, Thomas, Axel Korthaus, and Michael Rosemann (2009). "Business and Software Service Lifecycle Management." In: IEEE, pp. 87–96. DOI: 10.1109/EDOC.2009.20.

Krintz, Chandra, Hiranya Jayathilaka, Stratos Dimopoulos, Alexander Pucher, Rich Wolski, and Tevfik Bultan (2014). "Cloud Platform Support for API Governance." In: IEEE, pp. 615–618. DOI: 10.1109/IC2E.2014.90.

Leimeister, Jan Marco (2014). *Collaboration Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-20891-1.

Levin, Guy (2016). *An API-First Development Approach*. An API-First Development Approach. URL: https://dzone.com/articles/an-api-first-development-approach-1 (accessed on Aug. 24, 2018).

Lufthansa Group (2018). *Lufthansa API Portal*. URL: https://developer.lufthansa.com (accessed on Sept. 24, 2018).

Macvean, Andrew, Martin Maly, and John Daughtry (2016). "API Design Reviews at Scale." In: *CHI EA '16 Proceedings of the 2016 CHI Conference Extended Abstracts on*

Human Factors in Computing Systems, pp. 849–858. URL: http://dl.acm.org/ft_gateway.cfm?id=2851602&ftid=1716532&dwn=1&CFID=783074184&CFTOKEN=67606185.

Malinverno, Paolo and Mark O'Neill (2016). *Magic Quadrant for Full Life Cycle API Management*. Reprinted Report. Revised Version from Nov 30, 2016. Gartner.

Malinverno, Paolo and Mark O'Neill (2018). *Magic Quadrant for Full Life Cycle API Management*. Reprinted Report. Revised Version from 1 May, 2 018. Gartner. URL: https://www.gartner.com/doc/reprints?id=1-4Y21K37&ct=180501&st=sb.

Masse, Mark (2012). *REST API Design Rulebook*. 1st ed. Sebastopol: O'Reilly Media.

McBride, Gary (2017). *The Role of SOA Quality Management in SOA Service Lifecycle Management*. URL: https://www.ibm.com/developerworks/rational/library/mar07/mcbride/mcbride-pdf.pdf (accessed on Mar. 22, 2018).

Morgan, Jacob, Martin Gill, Randy Heffner, Alexander Causey, and Rachel Birell (2016). *Four Ways APIs Are Changing Banking - How Financial Services Firms Are Exploiting The API Economy*. Report. Forrester Research.

Mulesoft (2014). *Secrets of a Great API - Core principles for delivering successful APIs*. Whitepaper. Mulesoft. URL: https://www.mulesoft.com/lp/whitepaper/api/secrets-great-api (accessed on Mar. 13, 2018).

Murphy, Lauren, Tosin Alliyu, Mary Beth Kery, Andrew Macvean, and Brad A. Myers (2017). "Preliminary Analysis of REST API Style Guidelines." In:

Myers, Brad A. and Jeffrey Stylos (2016). "Improving API usability." In: *Communications of the ACM* 59.6, pp. 62–69. DOI: 10.1145/2896587.

Nasa (2018). *Nasa API Portal*. URL: https://api.nasa.gov/ (accessed on Sept. 24, 2018).

Newman, Sam (2015). *Building Microservices*. 1st. O'Reilly Media, Inc.

Nordic Apis (2016). *The API Economy*. URL: https://nordicapis.com/api-ebooks/the-api-economy/ (accessed on Apr. 18, 2018).

Parsons, Rebecca et al. (2017). *Technology Radar Vol. 17 - Insights into the technology and trends shaping the future*. Report 17. ThoughtWorks. URL: https://d1bss3tv0zspu7.cloudfront.net/assets/technology-radar-vol-17-en.pdf (accessed on Apr. 6, 2018).

Patni, Sanjay (2017). *Pro RESTful APIs: design, build and integrate with REST, JSON, XML and JAX-RS*. 1st ed. DOI 10.1007/978-1-4842-2665-0. Santa Clara, CA: Apress. 136 pp.

Pollard, Carol E., Dhiraj Gupta, and John W. Satzinger (2009). "Integrating SDLC and ITSM to 'Servitize' Systems Development." In: *AMCIS 2009*.

Postman (2017). *2017 Postman Community Report*. URL: http://pages.getpostman.com/rs/067-UMD-991/images/Postman-Survey-Results-Infographic.pdf (accessed on Aug. 16, 2018).

Programmable Web (2018). *Programmable Web*. URL: https://www.programmableweb.com/ (accessed on Sept. 24, 2018).

Ravichandran, Aruna, Kieran Taylor, and Peter Waterhouse (2016). *DevOps for Digital Leaders*. Berkeley, CA: Apress. DOI: 10.1007/978-1-4842-1842-6.

Rivero, José Matías, Sebastian Heil, Julián Grigera, Martin Gaedke, and Gustavo Rossi (2013). "MockAPI: An Agile Approach Supporting API-first Web Application Development." In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Red. by David Hutchison et al. Vol. 7977. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 7–21. DOI: 10.1007/978-3-642-39200-9_4.

Rodríguez, Carlos, Marcos Baez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali, and Gianraffaele Percannella (2016). "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices." In: *Web Engineering*. Ed. by Alessandro Bozzon, Philippe Cudre-Maroux, and Cesare Pautasso. Vol. 9671. Cham: Springer International Publishing, pp. 21–39. DOI: 10.1007/978-3-319-38791-8_2.

Sandhu, Ravi S. (1998). "Role-based Access Control." In: *Advances in Computers*. Vol. 46. Elsevier, pp. 237–286. DOI: 10.1016/S0065-2458(08)60206-5.

Schepers, T. G. J., M. E. Iacob, and P. A. T. Van Eck (2008). "A lifecycle approach to SOA governance." In: ACM Press, p. 1055. DOI: 10.1145/1363686.1363932.

Smartbear (2018). *Smartbear API Management*. URL: https://smartbear.com/ (accessed on Sept. 24, 2018).

Smith, Tom (2018). *DZone Research: API Management Issues*. DZone Research: API Management Issues. URL: https://dzone.com/articles/dzone-research-api-management-issues (accessed on Aug. 24, 2018).

Souza, Cleidson R. B. de, David Redmiles, Li-Te Cheng, David Millen, and John Patterson (2004). "How a good software practice thwarts collaboration: the multiple roles of APIs in software development." In: ACM Press, p. 221. DOI: 10.1145/1029894.1029925.

Spichale, Kai (2017). *API-Design: Praxishandbuch für Java- und Webservice-Entwickler*. Korrigierter Nachdruck. OCLC: 1004312491. Heidelberg: dpunkt.verlag. 347 pp.

Stafford, Jan (2018). *Why use new lifecycle tools in API management platforms?* Why use new lifecycle tools in API management platforms? URL: https://searchmicroservices.techtarget.com/feature/Why-use-new-lifecycle-tools-in-API-management-platforms (accessed on Aug. 24, 2018).

Stählin, Johannes, Sebastian Lang, Fabian Kajzar, and Christian Zirpins (2018). "Consumer-Driven API Testing with Performance Contracts." In: *Advances in Service-Oriented and Cloud Computing*. Ed. by Alexander Lazovik and Stefan Schulte. Vol. 707. Cham: Springer International Publishing, pp. 135–143. DOI: 10.1007/978-3-319-72125-5_11.

Tan, Wei, Yushun Fan, Ahmed Ghoneim, M. Anwar Hossain, and Schahram Dustdar (2016). "From the Service-Oriented Architecture to the Web API Economy." In: *IEEE Internet Computing* 20.4, pp. 64–68. DOI: 10.1109/MIC.2016.74.

Tan, Wui-Gee, Aileen Cater-Steel, and Mark Toleman (2009). "Implementing IT Service Management: A Case Study Focussing on Critical Success Factors." In: *Journal of Computer Information Systems* 50.2, pp. 1–12. DOI: 10.1080/08874417.2009.11645379.

Tech Target (2018). *How to smartly manage APIs through their full lifecycle*.

The W. Edwards Deming Institute (2018). *PDSA Cycle*. URL: https://deming.org/explore/p-d-s-a (accessed on Sept. 24, 2018).

Vasudevan, Keshav (2017). *How Bonotel Improved Collaboration and Accelerated API Delivery By Switching to SwaggerHub*. SwaggerBlog. URL: https://swagger.io/blog/api-strategy/switching-to-swaggerhub/ (accessed on Sept. 24, 2018).

Vester, John (2017). *RESTful API Lifecycle Management*. URL: https://dzone.com/storage/assets/4960646-dzone-rc238-restfulapilifecyclemanagement.pdf (accessed on Mar. 21, 2018).

Vukovic, Maja et al. (2016). "Riding and thriving on the API hype cycle." In: *Communications of the ACM* 59.3, pp. 35–37. DOI: 10.1145/2816812.

W3C (2018). *W3C Glossary*. URL: https://www.w3.org/TR/ws-gloss/ (accessed on Sept. 24, 2018).

Wähner, Kai (2014). *API Management as a Game Changer for Cloud, Big Data and IoT: Product Comparison and Evaluation*. Voxxed. URL: https://www.voxxed.com/2014/12/api-management-game-changer-big-data-cloud-iot-product-comparison-evaluation/ (accessed on Sept. 24, 2018).

Whitehead, Jim (2007). "Collaboration in Software Engineering: A Roadmap." In: IEEE, pp. 214–225. DOI: 10.1109/FOSE.2007.4.

Yin, Robert K. (2009). *Case Study Research: Design and Methods (Applied Social Research Methods)*. 4th ed. Sage Publications.

Zhang, Yajun, Jinlong Zhang, and Jiangtao Chen (2013). "Critical Success Factors in IT Service Management Implementation: People, Process, and Technology Perspectives." In: IEEE, pp. 64–68. DOI: 10.1109/ICSS.2013.38.

Appendices

A. Camunda

Camunda is a BPM tool which supports BPMN, CMMN and DMMN diagrams. A combination of the three diagram types is also possible and provides flexibility. Camunda was initially used for the automation approach, but later abandoned for this thesis, due to limited time reasons. Further research might be helpful and could use the presented results as basis.

A.1. BPMN

The BPMN diagram, shown in Figure A.1, depicts an initial version for using it in combination with the Camunda BPM tool. By using the Camunda Modeler[1], it is possible to connect code pieces to the diagram.

[1] https://camunda.com/products/modeler/

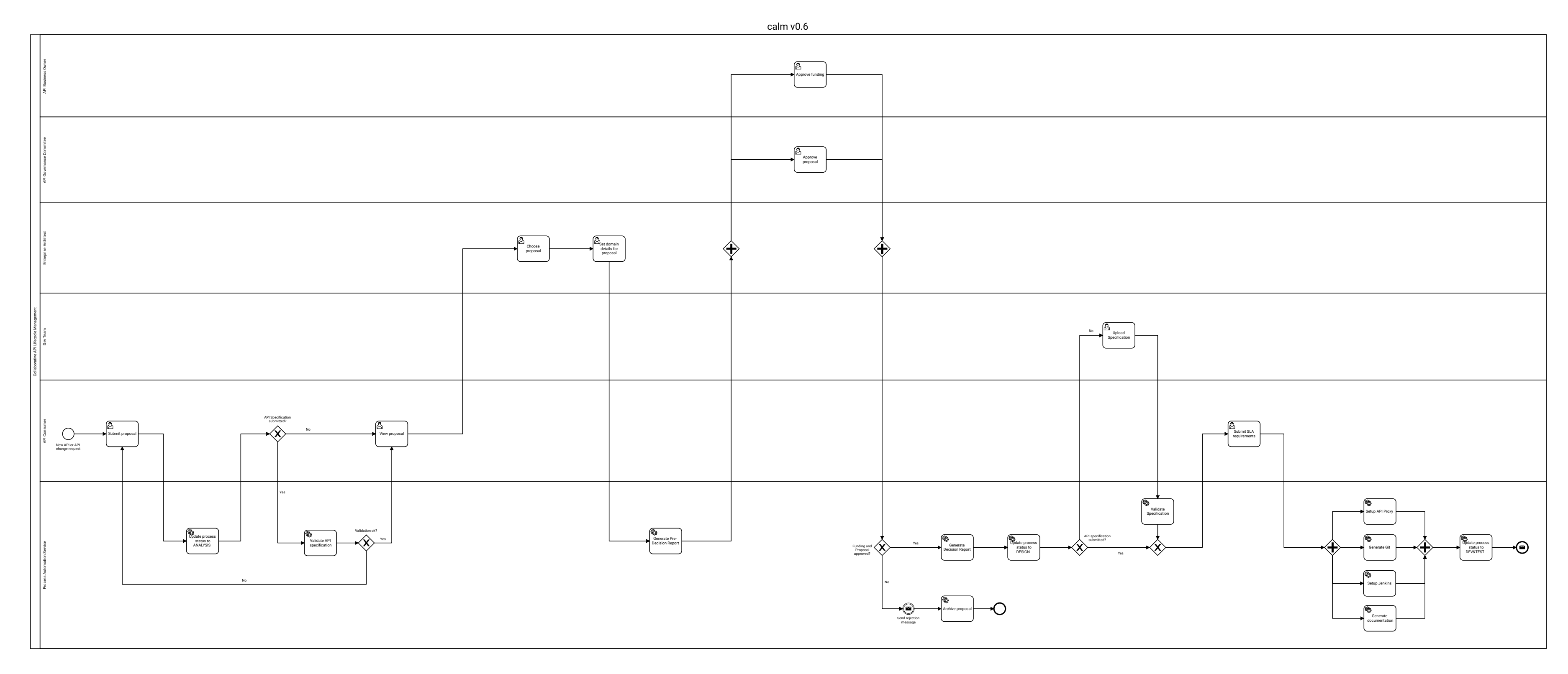


Figure A.1.: A BPMN approach for the prototype to support the Collaborative API LIfecycle Management

A.2. CMMN

The BPMN diagram does not provide flexibility. To overcome this hurdle, a CMMN was subsequently modeled. This CMMN diagram, depicted in Figure A.2, needs to be further extended. Automated work flows could be modeled with BPMN and combined with this CMMN diagram.

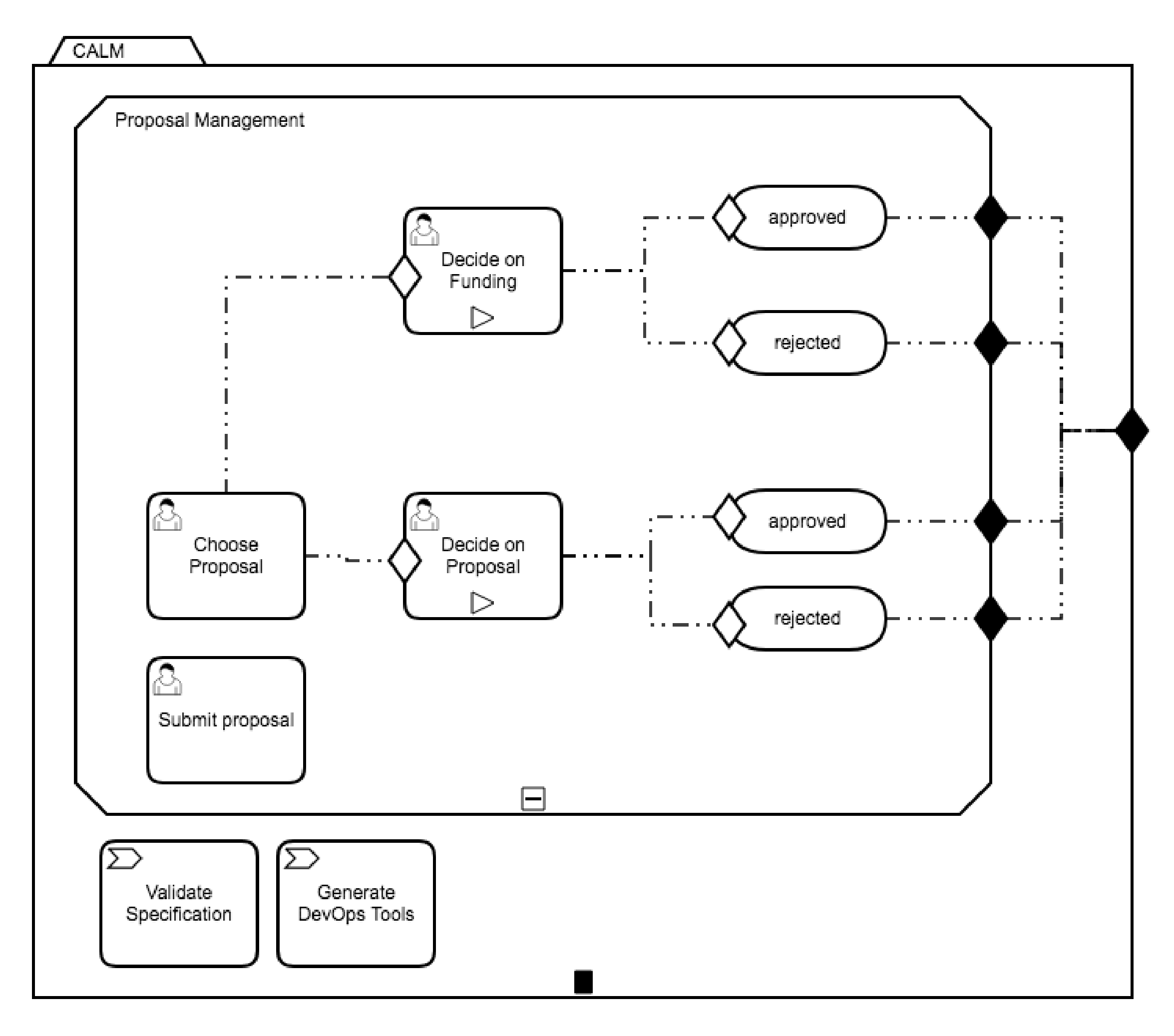


Figure A.2.: A BPMN approach for the prototype to support the Collaborative API Lifecycle Management

B. Evaluation Questionnaire

The evaluation was conducted with the following evaluation script. The evaluation script consists of two scenarios with several tasks. After each task the interviewees are asked to rate their perception of the prototype with the help of the SUS questionnaire. After both scenarios, the evaluation ends with four open qualitative questions.

Date: Time: ID:

Interviewee:

Job Title:

Years active in the current role:

Current Core Area:

Software Development Experience:

Evaluation – CALM Web App

Introduction

Given is a prototype in form of a web application from an European insurance company which lead people through the collaborative API Lifecycle Management process.

In the following, you are given one or two scenarios that require you to use the web application prototype to fullfill several tasks. After finishing all tasks, you are given a questionnaire that needs to be filled out by you.

Finally, there are a few open questions that the interviewer would like to ask you. Feel free to answer the questions with any opinion you might have.

Important Terms

Please ask the interviewer for clarification, if you do not understand the following terms:

- Web API
- API Specification



Date: Time: ID:

Evaluation Scenario 1 (API Consumer | Proposal Submission)

You are a financial technology startup, named *Ency*, who would like to have an API from the European insurance company, so that you can use that API to offer and sell health insurance products from them.

The following tasks cover your activities to get your wished API.

Task 1: Search for an existing suitable API proposal.

Your purpose is to get an API that enables you to sell specifically health insurance products. Your first impression is to search for an API proposal that already does, what you are looking for.

Firstly, get familiar with the User Interface. After that, look for an API proposal that also wants to sell specifically health insurance products. In case, you have found one, you would leave a note to that proposal that you are looking forward to use it too.

Task 2: Propose a new API.

In case, you have not found an existing suitable API proposal, your next intuition will be that you need to submit a new API proposal for your purpose of selling health insurance products.

Start to create a new proposal with the following details and submit it:

You are from the financial technology startup *Ency*. You want to have a *Health Insurance Products API* which *sells health insurance products* from the European insurance company. Your API proposal belongs to the *Insurance* category and has keywords like *Products, Health* and *Health Insurance*. The timeline for this API program should be *from 01/10/2018 until 01/04/2019*. For your API, you are willing to pay *3€ per request*.

Task 3: Improve your proposal by uploading some mock-up files.

Your proposal needs more details from your side. That is why, you are trying to provide more information, so that the reviewers can understand your purpose of your API better.

Look for your submitted proposal and upload the following mock-up file:
-> Mock-up file: “mockups_api_ency.pdf”

Task 4: Add some SLA requirements.

Additionally, you want to have certain standards for your new API to be fullfilled.

For that, you want to add the following two SLA requirements:
A) Availability of 99%
B) Uptime of 99%

Task 5: Upload your API specification and save your changes.

You already have a first idea, how the API could look like and what endpoints it should provide. For that, you created a first draft of an API specification for the new API and you want to attach it to the proposal.

Look for your proposal, upload the API specification and make necessary changes to save your updated API proposal.
-> API specification name: “api-spec_ency.yaml” or “api-spec_ency.json”

Task 6: Vote for your API.

Lastly, vote for your own API, so that it gets higher attention by the reviewers.

– Evaluation Scenario 1 done –

Date: Time: ID:

Evaluation Questionnaire – Scenario 1

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Date: Time: ID:

Evaluation Scenario 2 (API Provider | Review Proposals)

You are an enterprise architect with an insurance domain. You are in charge of screening proposals which are from the same domain as your own. In case of accepting an API proposal, the project will be automatically prepared for development.

The following tasks cover your activities for reviewing proposals and getting an automatic development setup.

Task 1: Search for all proposals with an insurance category that do not fullfill the given criteria and reject them.

Your task is to assess, if a proposal fullfills certain criteria to be accepted. For that, you need to screen all proposals that are on your review list and make a decision if the proposals fullfill certain requirements.

Screen through all proposals that need to be reviewed and reject all proposals that do not fullfill the following requirement:

-> above 2€ per request

Task 2: Approve the proposal with the highest votes.

Proposals with high votes attract your attention, because you know that other people are also interested in that API.

Look for the proposal from your domain with the highest votes. The proposal with the highest votes fullfills all basic requirements. That is why you will approve it.

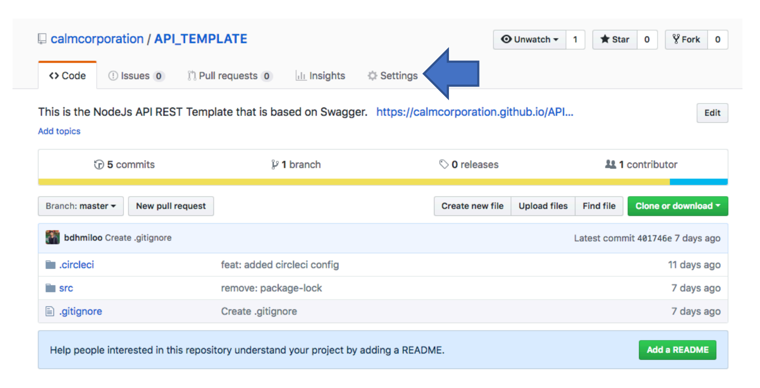
Task 3: Check out the generated project in github and prepare the API proposal for development.

By approving a proposal, the project will be automatically set up and you need to prepare to hand the proposal over for development.

Look for the approved proposal with the highest votes. Go to the repository and have a look at the code.

View the proposals' API documentation. Before you can do that, you need to initially enable it. The following instructions need to be executed, in order to enable the live documentation:

A) Go to the repository and click on the tab *Settings*.



– Please turn to next page –

Date: Time: ID:

B) Scroll down and look for the *Github Pages* section. Set the *Source* to *master branch /docs folder* and finally, click on the button *Save*.

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Source

GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. Learn more.

master branch /docs folder ▾ Save

Select source

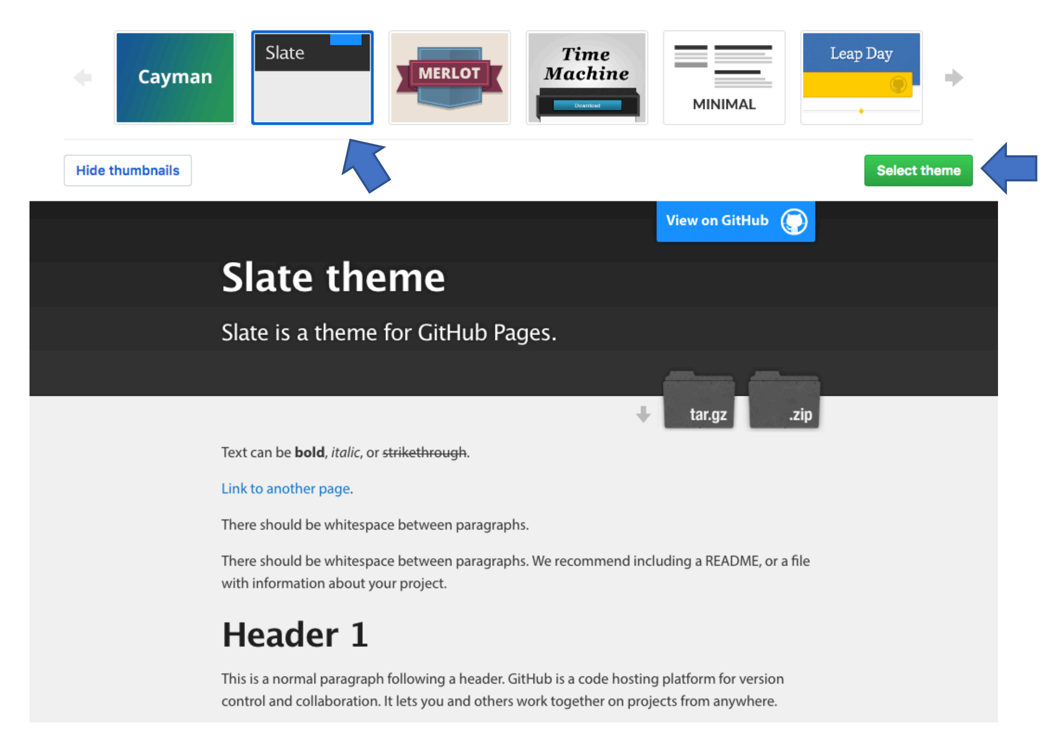
master branch
Use the master branch for GitHub Pages.

✓ master branch /docs folder
Use only the /docs folder for GitHub Pages.

None
Disable GitHub Pages.

...ll theme using the master branch /docs folder. Learn more.

C) Scroll down and look again for the *Github Pages* section. Click on the button *Choose a theme*. Choose for example *Slate* as documentation theme and click on the button *Select theme*.



D) Return to the web application prototype and have a look at the documentation for that proposal.

– Evaluation Scenario 2 done –

Date: Time: ID:

Evaluation Questionnaire – Scenario 2

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Date: Time: ID:

General Questions

1. Do you think the given web application could help to accelerate the API Lifecycle Management process? Explain why?

Answer:

2. Do you think the given web application facilitates the collaboration between all participating stakeholders? Explain why?

Answer:

3. Do you see any limitations or problems about the web application?

Answer:

4. Finally, what did you like about the web application?

Answer: